ABBUC EDITION
Ausgabe 2011

SYBEX

Das ATARI Profibuch

Julian Reschke/Andreas Wiethoff



Bearbeitungsstand: 24. Dezember 2011

Danksagung

Wir danken dem Sybex-Verlag und den Autoren Julian Reschke & Andreas Wiethoff für die freundliche Genehmigung, den Inhalt dieses Standardwerks den Freunden der ATARI-8-Bit-Computer frei zur Verfügung stellen zu dürfen.

Herten, im Dezember 2010

Hinweise

Bei der digitalen Aufarbeitung des Originals wurden die Regeln der neuen Rechtschreibung angewendet. Bilder und Screenshots wurden in Farbe erneuert.

Die Reihe der 8-Bit-Computer war bis 1992 weitergeführt worden. Die sich daraus ergebenden Informationen wurden zusammen mit inhaltlich erforderlichen Korrekturen und Ergänzungen eingefügt.

Vorschläge für Änderungen bitte an ABBUC e.V. senden.

Ein Projekt des ABBUC e.V., ermöglicht durch:

patjomki
mp-one
Eda70
mega-hz
Hias
Bernd
Dietrich
www.atari-computermuseum.de
Sleepy
skriegel
tfhh
andreasb
des-or-mad
Carsten
Rockford
GoodByteXL

Das ATARI Profibuch

Julian Reschke

Andreas Wiethoff

Erweiterte und ergänzte Auflage
ABBUC e.V. (P) 2011

Anmerkungen in der 1. Auflage von 1985:

Bundesliga, CAVELORD und Memo-Box sind Warenzeichen von AXIS Komputerkunst.

ATARI 400, 600XL, 800, 800XL, 810, 815, 825, 1050, 1200XL, 1400XL, 1450XLD und 130XE sind Warenzeichen von ATARI Corp.

HIGHWAY-DUEL ist ein Warenzeichen von DYNAMICS Marketing GmbH.

M.U.L.E. ist ein Warenzeichen von Electronic Arts.

DOS XL, MAC/65, ACTION!, BASIC XL und The Writer's Tool sind Warenzeichen von Optimized Systems Software.

NADRAL ist ein Warenzeichen von Julian Reschke & Andreas Wiethoff.

Bildschirmfotos mit freundlicher Genehmigung der Firmen AXIS Komputerkunst und DYNAMICS Marketing.

Umschlagentwurf: Daniel Boucherie/tgr
Satz: tgr — typo-grafik-repro gmbh, Remscheid
Gesamtherstellung: Boss Druck und Verlag, Kleve

ISBN 3-88745-605-X (ungültig für die ABBUC-Ausgabe)
1. Auflage 1985

Ergänzung durch ABBUC e.V. (P) in 2010/2011:

ATARI Corporation und OSS existieren nicht mehr.
ICD Inc. hatte die Produktlinie für ATARI-8-Bit-Computer an FTe verkauft; FTe existiert nicht mehr.

Wichtige Hard- und Software, die erst nach dem Erscheinen dieses Buchs auf den Markt kam, wurde neu hinzugefügt.

Produktion und Vertrieb der ATARI-8-Bit-Computer wurde 1992 eingestellt.

ABBUC-Logo - http://des-or-mad.net

Inhaltsverzeichnis

Vorwort 1985

Die letzten Dateien sind ausgedruckt, alle Fotos in letzter Minute entwickelt, vergrößert, getrocknet, geschnitten und gepresst, das Waten durch die stetig wachsenden Papierberge wird immer mühsamer, Drucker und Diskettenstationen sind kurz vorm Qualmen (ebenso wie unsere Köpfe) - wir sind fertig (in dem einen wie in dem anderen Sinne).

Doch dieses für den passionierten und engagierten Programmierer gewohnte Chaos soll demnächst der Vergangenheit angehören, denn das ATARI-Profibuch enthält alle Informationen, die Sie jemals beim Programmieren brauchen werden!

Schließlich hat uns während der gesamten Arbeitszeit kein Bestandteil unseres Geräteparks im Stich gelassen:

- 2 ATARI 800
- 1 ATARI 800XL
- 2 ATARI 130XE
- 3 ATARI 810
- 5 ATARI 1050
- 1 ATARI 825
- 1 Epson FX 80+ (eine Leihgabe des Verlages)
- 1 Panasonic KX P-1092, der innerhalb eines Tages ohne zu murren das gesamte Manuskript ausdruckte
- 4 Datensichtgeräte
- 1 Pentax MX
- 1 Ricoh XR-S

Bei unserer Arbeit haben uns tatkräftig unterstützt:

Sebastian Klose, der für die Fotos verantwortlich zeichnet,
Ilse Klose mit der Bereitstellung von Requisiten,
Michael Mannl, der für Papier sorgte,
die Herren Grikscheit und Lehmann von der Firma ATARI Corp.,
Dieter Hegemann, Informationen aus erster Hand,
Werner Breuer mit viel Geduld,
Rainer Lüers mit regelmäßigem Motivationsnachschub,
Gabriele Wentges und Norbert Hesselmann von SYBEX.

Ohne folgende LPs wäre es nicht auszuhalten gewesen:

KIM WILDE - Teases & Dares
STING - The Dream of the Blue Turtles
DIRE STRAITS - Brothers in Arms

SUPERTRAMP - Brother Where You Bound
WORKING WEEK - Working Nights
TANGERINE DREAM - Logos

Gewidmet ist dieses Buch:

Allen, die ausgerechnet an ihrem Geburtstag besonders hart arbeiten mussten.

Allen, die von ihrem ATARI-Computer genauso überzeugt sind wie wir.

Sebastian Klose, Julian Reschke, Andreas Wiethoff

Vorwort 2011

Als im Frühling 2010 die Freigabe wahr wurde, war ein Entschluss schnell gefasst: Das ist eine Aufgabe für den ABBUC!

Die anfängliche Idee, das Original von 1985 einfach in digitaler Form zur Verfügung zu stellen, erwies sich bald als nicht tragfähig, da nach 1985 noch einiges an Hardware und Software bei ATARI erschienen war. Und damit nicht genug. Bis heute erscheinen fast schon regelmäßig neue Hard- und Software für den "kleinen ATARI", was die Szene belebt. Auch war manche Information aus dem originalen Werk überholt oder fehlte gänzlich. So wurde aus einer Freizeitsache schnell ein anspruchsvolles, zeitintensives Projekt, das nur durch gute Zusammenarbeit im ABBUC e.V. auf den derzeitigen Stand zu bringen war.

Schwierig war und ist es, den Charakter des ursprünglichen Profibuchs trotz der Korrekturen, Ergänzungen und Änderungen beizubehalten. Die Versuchung war groß, daraus ein ATARI-8-Bit-Kompendium zu entwickeln, gibt es doch noch so viel mehr an Information. Die nun vorliegende Fassung stellt dahingehend einen Kompromiss dar, der nicht zuletzt den Autoren des zugrunde liegenden Ur-Profibuchs geschuldet ist.

Vom 1. Mai bis 22. Dezember 2010 flossen unzählige Stunden in dieses Projekt ein. Im Namen aller Mitstreiter und Unterstützer danke ich den Autoren und dem Verlag für die Freigabe – mögen ihr noch einige folgen, damit wir weiterhin viel Spaß mit dem "kleinen ATARI" erleben.

Nach der digitalen Veröffentlichung zu Weihnachten 2010 auf der Webseite des ABBUC e.V. gingen eine ganze Reihe an Fehlermeldungen, Verbesserungsvorschlägen und Ergänzungshinweisen ein. Diese sind nun ebenso wie eine Reihe an Korrekturen im Layout in die Überarbeitung dieses Standardwerkes eingeflossen. Danke an alle, die entsprechende Hinweise gegeben, Hilfestellung oder Zuarbeit geleistet haben.

Mein besonderer Dank gilt 'Dietrich' für die gründliche Qualitätskontrolle und 'Sleepy' für das unermüdliche Fotografieren.

Den Beginn dieses Projekts haben wir 'patjomki' zu verdanken, der die Freigabe erwirkt und zur rechten Zeit den Anstoß gegeben hatte.

GoodByteXL

1 Speicherplan

Sofern nicht anders angegeben, beziehen sich sämtliche Erklärungen auf alle ATARI-Computer. Abschnitte, die nur für den ATARI 400 und 800 gelten, sind durch (A); Abschnitte, die nur für XL/XE-Geräte Gültigkeit haben, sind durch (X) markiert. Dies wird in allen Kapiteln beibehalten.

1.1 Seite 0 (Page Zero)

Die erste Hälfte der für den 6502-Prozessor des ATARI so wichtigen Page 0 ist für das Betriebssystem reserviert, die zweite Hälfte wird von Anwenderprogrammen wie BASIC benutzt.

0,1 $0,$1 LINZBS (A)
Diese Speicherstelle ist nur während des Initialisierungsprozesses (Kalt- oder Warmstart) von Bedeutung.

0 $0 LINFLG (X)
Wird vom Betriebssystem nicht benutzt und weder bei Kalt- noch Warmstart (Reset) gelöscht.

1 $1 NGFLAG (X)
Dieses Flag wird während der Speicherüberprüfung benutzt, um RAM- oder ROM-Fehler zu registrieren. Dazu wird es zu Beginn des Bootprozesses auf den Wert 1 gesetzt, und jedes Mal, wenn ein Fehler entdeckt wird, um ein Bit nach rechts verschoben, sodass am Ende des Kaltstarts eine 0 das Auftreten eines Speicherfehlers signalisiert.

Beim Reset werden 1, 4-6, 8 und 16-127 initialisiert, der Rest von Page 0 bleibt unverändert.

2,3 $2,$3 CASINI
Durch diese Adresse wird bei der Initialisierung eines Kassettenprogrammes gesprungen. Sie stammt aus dem ersten Record (jeder Record umfasst 128 Bytes) eines Kassettenfiles, dessen erste sechs Bytes folgende spezielle Bedeutung haben: Das erste Byte wird ignoriert, das zweite Byte gibt die Anzahl der noch zu ladenden Kassettenrecords und das dritte und vierte Byte die Anfangsadresse an. Das fünfte und sechste Byte wird nach CASINI übertragen. Der Initialisierungsprozess läuft wie folgt: Nach dem Laden der erforderlichen Records wird durch die Ladeadresse plus sechs gesprungen, um ein weiteres Laden zu ermöglichen. Dies muss dann aber durch ein spezielles Programm an dieser Stelle ausgeführt werden. Nach einem CLC + RTS wird zur Initialisierung durch den Vektor CASINI gesprungen (SEC + RTS bewirkt die Anzeige "BOOT ER-

ROR" und Wiederholung des Boot-Prozesses). Zum Abschluss des Bootvorgangs wird an die durch DOSVEC (10,11; $A,$B) angegebene Adresse verzweigt.

Am Ende der Reset-Routine wird ebenfalls durch CASINI gesprungen, sofern Bit 1 von BOOT? (9; $9) gesetzt ist.

4,5 $4,$5 RAMLO

Diese Speicherstelle ist sowohl ein Zeiger, der während des Einschaltvorgangs benutzt wird, als auch die Adresse für die Fortsetzung des Disketten- oder Kassettenbootvorgangs (dies ist die Ladeadresse plus sechs).

6 $6 TRAMSZ (A)

Dieses Register hat den Wert 1, wenn eine Cartridge (beim ATARI 800 ist dies die linke) eingesteckt ist.

6 $6 TRNSMZ (X)

Temporäres Register für die RAM-Speichergröße beim Reset. Wird am Ende der Reset-Routine auf 1 gesetzt, falls ein Modul oder das BASIC aktiv ist, und andernfalls auf 0.

7 $7 TSTDAT (A)

Wert 1 in diesem Register bedeutet, dass das rechte Modul eingesetzt ist.

7 $7 TSTDAT (X)

Wird vom Betriebssystem nicht benutzt und weder beim Kalt- noch Warmstart (Reset) gelöscht.

8 $8 WARMST

Während des Kaltstarts enthält dieses Register den Wert 0, nach dem ersten Drücken von RESET oder Aufruf der Reset-Routine den Wert 255.

9 $9 BOOT?

Bit 0 wird vom Betriebssystem gesetzt, wenn von Diskette gebootet wurde, Bit 1, wenn von Kassette gebootet wurde. Ist beim Drücken von RESET Bit 0 gesetzt, wird der Vektor DOSINI (12,13; $C,$D) aufgerufen, ist Bit 1 gesetzt, CASINI (2,3; $2,$3).

10,11 $A,$B DOSVEC

Am Ende der Reset-Routine springt das Betriebssystem durch diesen Vektor, sofern weder ein Modul noch das BASIC aktiv ist. Beim Booten eines Programms von Diskette oder Kassette legt dieses meist seine Startadresse hier ab. Z.B. ist dies bei einem gebooteten ATARI DOS 2.5 die Startadresse des DUP (Disk Utility Package). Der BASIC-Befehl "DOS" springt ebenfalls durch diesen Vektor und ruft damit das DUP auf.

12,13 $C,$D DOSINI

Diskettenprogramme werden über diesen Vektor durch ein JSR-Kommando initialisiert. Siehe auch BOOT? (9; $9). DOSINI funktioniert für Bootdisketten genauso wie CASINI (2; $2) für Bootkassetten, siehe dort.

14,15 $E,$F APPMHI

Dies ist ein Zeiger auf die Adresse, von der an aufwärts die Bildschirmdaten und die Display List liegen können. Für das Betriebssystem wird durch diese 2 Bytes die Adresse angegeben, bis zu der sich ein Benutzerprogramm erstrecken kann. Falls ein Einschalten einer Grafikstufe (GRAPHICS-Befehl in BASIC; OPEN-Befehl für den Bildschirm über die CIO in Maschinensprache) es erforderlich machen würde, dass Bildschirmdaten bzw. die Display List unterhalb der in APPMHI angegebenen Adresse liegen, so wird dieser Befehl nicht ausgeführt, sondern statt dessen ein Fehler angezeigt und die Grafikstufe 0 eingeschaltet.

Alle folgenden Register werden nicht nur beim Kaltstart, sondern auch bei jedem Warmstart initialisiert.

16 $10 POKMSK

Diese Speicherstelle ist das Schattenregister für IRQEN (53774; $D20E). Die einzelnen Bits dieses Registers ermöglichen bzw. sperren die verschiedenen IRQ-Interrupts (Bit gesetzt: Interrupt möglich). Siehe auch Hinweise vor der Adresse VDSLST (512,513; $200,$201).

Bit	Dez	Hex	Funktion
7	128	$80	Die Breaktaste ist wirksam.
6	64	$40	Der Tastaturinterrupt ist ermöglicht.
5	32	$20	Der "serielle-Dateneingabe-fertig" - Interrupt ist ermöglicht.
4	16	$10	Der "serielle-Datenausgabe-benötigt"-Interrupt ist ermöglicht.
3	8	$8	Der "serielle-Datenausgabe-beendet"-Interrupt ist ermöglicht.
2	4	$4	Der Interrupt des vierten POKEY-Timers ist ermöglicht. (POKEY-Timer 4 wird nur vom XL- und XE-Betriebssystem unterstützt.)
1	2	$2	Der Interrupt des zweiten POKEY-Timers ist ermöglicht.
0	1	$1	Der Interrupt des ersten POKEY-Timers ist ermöglicht.

Eine Sperrung der BREAK-Taste ist durch Ausmaskieren von Bit 7 in diesem Register und im zugehörigen Hardwareregister IRQEN (53774; $D20E) möglich. Durch jeden PRINT- oder OPEN-Befehl auf den Bild-

schirm oder auf den Editor wird dieses Bit wieder gesetzt, sodass die BREAK-Taste wieder Wirkung zeigt. Es muss also nach jedem PRINT- oder OPEN-Befehl wieder ausmaskiert werden. Der Begriff Timer-Interrupt bedeutet, dass die mit der entsprechenden Nummer versehenen AUDF-Register (ab 53760; $D200) als Timer benutzt werden. Wenn diese Register nach Herunterzählen von einem durch den Benutzer vorgegebenen Wert 0 erreicht haben, wird ein Interrupt ausgelöst und indirekt durch die zugehörigen VTIMR-Register (528-533; $210-$215) gesprungen. Die Zähler werden durch Einschreiben eines beliebigen Wertes in STIMER (53769; $D209) gestartet und rufen periodisch wiederkehrend die Interruptroutine auf. Die Timer müssen also nur einmal gesetzt und gestartet werden.

Die XL/XE-Geräte besitzen zusätzlich einen eigenen Vektor für die BREAK-Taste (BRKKY: 566,567; $236,$237).

17 $11 BRKKEY
Null: BREAK-Taste wurde gedrückt. Jede andere Zahl: BREAK wurde nicht gedrückt. Das Drücken von BREAK während einer I/O-Operation zieht einen Error 128 ($80) nach sich, sofern die I/O-Routine keine weiteren Wiederholversuche mehr macht. Siehe auch unter POKMSK (16; $10).

18,19,20 $12,$13,$14 RTCLOK
Zur einfachen Zeitmessung für den Benutzer besitzt der ATARI-Computer eine eingebaute, softwaremäßig gesteuerte Uhr.

Die Register 18, 19 und 20 werden als 3-Byte-Zähler alle 1/50 Sekunde (während jedes Vertikal-Blanks) um 1 erhöht. Register 20 erreicht dabei alle 5,1 Sekunden den Wert 255, Register 19 alle 21,845 Minuten und Register 18 alle 92,84 Stunden. Da diese Uhr vom Betriebssystem nicht benutzt wird und auch während zeitkritischer I/O-Operationen weiter erhöht wird, kann sie vom Benutzer ohne Probleme verwendet werden.

21,22 $15,$16 BUFADR
Wird von der DSKINV-Routine (58451; $E453) temporär als Zeiger auf den benutzten Diskettenpuffer benutzt.

23 $17 ICCOMT
Enthält beim CIO-Aufruf den Index auf die zu benutzende Treiberroutine in der zugehörigen Treibertabelle (0=open, 2=close, 4=read, 6=write, 8=status, 10=special).

Die nächsten 4 Bytes werden vom DUP des DOS (Versionen 2.0s und 2.5) für interne Zwecke verwendet. Wie das DUP aussieht (Menü, Kommandointerpreter, Filemanager) und ob es mit im DOS.SYS-File enthalten ist oder als eigenes File existiert, darüber entscheidet der Programmierer.

24,25 $18,$19 DSKFMS
Sprungvektor ins File Management System (FMS) in ATARI-DOS.

26,27 $1A,$1B DSKUTL
Startadresse des Disk Utility Package (DUP) in ATARI-DOS.

28 $1C PTIMOT (A)
Die Werte in diesem Register entsprechen in etwa dem Zeitraum in Sekunden, in dem der Drucker eine positive Statusmeldung an den Computer gesandt haben muss. Ansonsten wird I/O-Fehler 138 (Time Out!) gemeldet. Dieses Register wird durch die Druckertreibersoftware bei jedem STATUS-Befehl neu initialisiert. Bei den XL/XE-Geräten ist dieses Register nach 788 ($314) verlegt worden.

29 $1D PBPNT (A)
Zeiger auf das nächste zu übertragende Zeichen im Druckerpuffer (0 entspricht der Anfangsadresse des Puffers). Der Wert liegt zwischen 0 und dem Wert in PBUFSZ. Bei den XL/XE-Geräten ist dieses Register nach 734 ($2DE) verlegt worden.

30 $1E PBUFSZ (A)
Der Druckerpuffer umfasst so viele Bytes (Zeichen), wie dieses Register angibt. Wenn PBPNT und PBUFSZ die gleiche Zahl enthalten, wird der Inhalt des Druckerpuffers an den Drucker übergeben. Die Größe des Puffers beträgt normalerweise 40 Zeichen. Bei den XL/XE-Geräten ist dieses Register nach 735 ($2DF) verlegt worden.

31 $1F PTEMP (A)
Temporäre Speicherstelle, die das Zeichen enthält, das gerade gedruckt werden soll.

28-31 $1C-$1F ABUFPT (X)
Diese 4 Bytes werden temporär von den sogenannten Gerätetreiberladeroutinen verwendet.

Da das Betriebssystem auch auf der Seite 0 einen I/O-Kontrollblock benötigt, stellen die Speicherstellen 32-47 ($20-$2F) den ZIOCB (Zero-Page-Input-Output-Control-Block) dar. Dieser I/O-Kontrollblock hat die gleiche Struktur wie die acht I/O-Kontrollblöcke, die ab 832 ($340) zu finden sind. Die Struktur wird im Einzelnen dort erklärt. Der ZIOCB kontrolliert die I/O-Operationen zwischen der CIO und den einzelnen Treibern. Bei Benutzung der CIO wird der jeweilige IOCB in den ZIOCB übertragen und mit diesem weitergearbeitet (siehe auch Abschnitt über Gerätetreiber).

Die einzelnen ZIOCB-Register:

32	$20	ICHIDZ
33	$21	ICDNOZ
34	$22	ICCOMZ
35	$23	ICSTAZ
36	$24	ICBALZ
37	$25	ICBAHZ
38	$26	ICPTLZ

39	$27	ICPTHZ
40	$28	ICBLLZ
41	$29	ICBLHZ
42	$2A	ICAX1Z
43	$2B	ICAX2Z
44,45	$2C,$2D	ICSPRZ

46 $2E ICIDNO
Nummer des IOCB multipliziert mit 16.

47 $2F CIOCHR
Bei Übertragung einzelner Bytes Zwischenregister für den betreffenden Wert.

Die Register 48-66 ($30-$42) sind die Speicherstellen auf der Seite 0 für die serielle Schnittstelle (SIO).

48 $30 STATUS
Statusregister, das nur interne Bedeutung hat.

49 $31 CHKSUM
Prüfsumme der gesendeten oder empfangenen Daten. Bei diesem Byte wird das Carry-Flag (Marker für Übertrag) in Bit 0 übertragen. Genauer: Die Daten werden zur Ermittlung der CHECKSUM fortlaufend addiert. Ist dabei eine Zwischensumme größer als 255, wird 255 davon abgezogen (in Maschinensprache: CLC, ADC byte, ADC #0).

50,51 $32,$33 BUFRLO/HI
Diese Speicherstelle zeigt auf das seriell zu übertragende Byte im Datenpuffer. Dieses Byte wird, falls gesendet wird, in SEROUT (53773; $D20D) übertragen; falls empfangen wird, aus SERIN (53773; $D20D) übernommen. SERIN speichert die acht Bits, die einzeln auf das externe Gerät übertragen bzw. von diesem empfangen werden.

52,53 $34,$35 BFENLO/HI
Adresse des ersten Bytes nach dem Puffer, auf den durch BUFRLO/HI gezeigt wird.

54 $36 CRETRY (A)
Höchstzahl der Wiederholversuche, einen Befehl an die verschiedenen externen Geräte zu geben. Das Register wird auf 13 ($D) initialisiert. Bei den XL/XE-Geräten ist dieses Register nach 668 ($29C) verlegt worden.

55 $37 DRETRY (A)
Höchstzahl der Wiederholversuche (normaler Wert ist 1) des externen Gerätes, einen Befehl auszuführen. Bei den XL/XE-Geräten ist dieses Register nach 701 ($2BD) verlegt worden.

54,55 $36,$37 LTEMP (X)
Zwischenregister für die Gerätetreiberladeroutinen.

Es folgen verschiedene Flags (Marker), die einen bestimmten Zustand nach einer Operation über den seriellen Bus (SIO-Operation) anzeigen.

56 $38 BUFRFL
255 bedeutet: Datenpuffer ist voll.

57 $39 RECVDN
255 bedeutet: Empfang ist beendet.

58 $3A XMTDON
255 bedeutet: Übertragung ist beendet.

59 $3B CHKSNT
255 bedeutet: Prüfsumme ist übertragen worden.

60 $3C NOCKSM
Jeder Wert ungleich 0 bedeutet, dass keine Prüfsumme nach der Datenübertragung folgt. Falls 0 in dieser Speicherstelle steht, folgt eine Prüfsumme der Datenübertragung.

61 $3D BPTR
Die Addition von CASBUF (1021; $3FD) und BPTR ergibt das gerade zu übertragende Byte für den Kassettenrecorder. Der Wert liegt zwischen 0 und dem Wert in BLIM (650; $28A).

62 $3E FTYPE
Der Wert in dieser Speicherstelle stammt aus DAUX2 (779;$30B) bzw. ICAX2 des betreffenden I/O-Kontrollblocks und gibt die Art des Zwischenraums zwischen einzelnen Kassettenrecords an. 0 bedeutet, dass normale Zwischenräume zwischen den Records erzeugt werden. Ein positiver Wert ungleich 0 bedeutet, dass nur kurze Zwischenräume zwischen den einzelnen Records eingefügt werden.

63 $3F FEOF
Wenn der Inhalt dieses Registers 0 ist, so ist das Ende eines Kassettenfiles noch nicht erreicht. Das Ende eines Kassettenfiles wird durch 254 ($FE) im Kommandobyte eines Kassettenrecords angezeigt.

64 $40 FREQ
Zahl der Pieptöne beim Öffnen eines Kassettenfiles (1: Laden; 2: Speichern).

65 $41 SOUNDR
Null bedeutet: keine Geräusche während I/O-Operationen. Jeder andere Wert bedeutet, dass die bekannten Lese- und Schreibgeräusche während der I/O-Operation erzeugt werden.

66 $42 CRITIC
Während I/O-Operationen wird dieses Register auf einen Wert ungleich 0 gesetzt. In diesem Fall wird der zweite Teil des System-Vertical-Blank-Interrupts nicht mehr ausgeführt. Hierin werden unter anderem verschiedene Zähler vermindert, Schattenregister in die entsprechenden Hardwareregister übertragen und die Wiederholfunktion der Tasten durchgeführt. Außerdem wird der sogenannte "Deferred VBI", der vom Anwender programmiert wird und auf den durch den Vektor VVBLKD (548,549; $224,$225) gezeigt wird, nicht mehr aufgerufen. Dieses Register sollte vom Benutzer nicht verändert werden.

67-73 $43-$49 FMSZPG
Verschiedene Register für DOS. Bei ATARI DOS 2.x sind dies:

67,68 $43,$44 ZBUFP
Diese beiden Bytes zeigen auf den Dateinamen, der vom Anwender angegeben wird.

69,70 $45,$46 ZDRVA
Diese beiden Speicherstellen werden von verschiedenen DOS-Routinen verwendet. Sie zeigen meist auf den VTOC-Puffer des aktuell genutzten Laufwerks.

71,72 $47,$48 ZSBA
Diese beiden Bytes zeigen auf den Sektorpuffer der gerade benutzten Datei.

73 $49 ERRNO
Dieses Register speichert die Nummer des aufgetretenen Fehlers ab. Eine Fehlercodetabelle ist im Tabellenteil zu finden.

74 $4A CKEY (A)
Während des Kaltstarts wird vom Computer die START-Taste abgefragt; falls sie gedrückt ist, wird CKEY auf einen Wert ungleich von 0 gesetzt und von der Kassette gebootet. Bei den XL/XE-Geräten ist dieses Register nach 1001 ($3E9) verlegt worden.

75 $4B CASSBT (A)
Falls erfolgreich von Kassette gebootet wurde, wird in dieses Register ein Wert ungleich von 0 geschrieben. Siehe auch BOOT? (9; $9). Bei den XL/XE-Geräten ist dieses Register nach 1002 ($3EA) verlegt worden.

74,75 $4A,$4B ZCHAIN (X)
Vektor für die Gerätetreiberladeroutinen.

76 $4C DSTAT
Diese Speicherstelle ist das Statusregister für den Bildschirm und die Tastatur. Hier werden alle Fehlercodes abgelegt, die den Bildschirmtreiber (Editor) betreffen.

77 $4D ATRACT
Der Atari-Computer verfügt über eine Schutzfunktion für den Bildschirm, die dafür sorgt, dass, falls ca. 11 Minuten lang keine Taste gedrückt wird, alle Farben auf einen dunklen Wert gesetzt werden, damit sich kein Bild in der Bildröhre einbrennt. Dieses Register wird im selben Zeitabstand wie Register 19 ($13), also alle 5,1 Sekunden, erhöht. Bei jedem Tastendruck wird es durch die Tastatur-Interrupt-Routine auf 0 gesetzt. Wenn diese Speicherstelle 128 ($80) erreicht, so wird sie auf 254 ($FE) gesetzt und das "Atracten" beginnt. Bei Programmen, die die Tastatur nicht verwenden, sollte dieses Register beispielsweise infolge einer Bewegung eines Steuerknüppels zurückgesetzt werden.

78 $4E DRKMSK
Im normalen Betrieb enthält dieses Register die Zahl 254 ($FE), im Attract-Modus jedoch 246 ($F6), damit die Farben des Bildschirms nicht zu hell werden.

79 $4F COLRSH
Im Attract-Modus werden alle Farbregister mit den Registern DRKMSK und COLRSH verknüpft (AND DRKMSK, EOR COLRSH), um die Helligkeit der Farben zu reduzieren. COLRSH enthält denselben Wert wie die Speicherstelle 19 ($13).

Die Speicherstellen 80-122 ($50-$7A) werden vom Editor und Bildschirmtreiber benötigt.

80 $50 TMPCHR
Zeichen, das gerade auf den Bildschirm geschrieben wird.

81 $51 HOLD1
Wie Register 80 ($50).

82 $52 LMARGN
Der Benutzer kann die linke und rechte Grenze des Textbildschirms beliebig festlegen. Dies geschieht durch das Schreiben von entsprechenden Werten in dieses und das folgende Register. Das Betriebssystem setzt diese Speicherstelle auf 2, da manche NTSC-Fernseher nicht die gesamte Breite des vom Computer erzeugten Bildes darstellen können.

83 $53 RMARGN
Rechte Grenze des Textbildschirms, die vom Betriebssystem auf 39 ($27) initialisiert wird.

84 $54 ROWCRS
Die augenblickliche Zeilenposition des Text- oder Grafikcursors wird in dieser Speicherstelle abgelegt; in der Grafikstufe 0 liegt sie zwischen 0 und 23 ($17), in Grafikstufe 8 zwischen 0 und 191 ($BF).

85,86 $55,$56 COLCRS
Momentane Spaltennummer des Text- oder Grafikcursors; da in Grafikstufe 8 Werte zwischen 0 und 319 möglich sind, werden 2 Bytes benötigt. In allen anderen Grafikstufen ist Register 86 ($56) jedoch immer 0.

87 $57 DINDEX
Dieser Wert wird aus dem GRAPHICS-Befehl entnommen oder, bei einem OPEN-Kommando, aus den vier unteren Bits (sog. low nibble) des AUX1-Bytes des betreffenden I/O-Kontrollblocks und gibt die im Augenblick eingeschaltete Grafikstufe an. Wenn man nun einen Wert für eine andere als die eingeschaltete Grafikstufe hineinschreibt, so "denkt" das Betriebssystem, es wäre die vorgetäuschte Grafikstufe eingeschaltet, und führt PLOT-, DRAWTO- und FILL-Befehle (XIO 18) anders aus.

88,89 $58,$59 SAVMSC
Dieses Register gibt die Adresse des Zeichens in der linken oberen Ecke des Bildschirms an und ist daher auch gleichzeitig ein Zeiger auf die niedrigste Adresse des Bildspeichers. Die Adresse des Textfensters (Grafikstufen 1-8) ist in TXTMSC (660,661; $294,$295) zu finden.

Die folgenden neun Register (90-98; $5A-$62) werden für den DRAWTO- und FILL-Befehl verwendet.

90 $5A OLDROW
Beim DRAWTO- oder FILL-Kommando wird die vorherige Zeilennummer des Text- bzw. Grafikcursors benötigt. Dieser Wert wird aus ROWCRS (84; $54) übernommen.

91,92 $5B,$5C OLDCOL
Dieses Register stellt die ehemalige Spaltennummer des Text- bzw. Grafikcursors dar, die beim DRAWTO- oder FILL-Kommando benötigt wird. Die Werte werden aus COLCRS (85,86; $55,$56) übernommen.

93 $5D OLDCHR
Das Zeichen, das zurzeit unter dem Cursor steht, ist vorher in diesem Register gespeichert worden. Wenn der Cursor bewegt wird, wird das Zeichen aus diesem Register auf seinen Originalwert zurückgesetzt.

94,95 $5E,$5F OLDADR
Diese Adresse zeigt auf die Stelle im Speicher, an der der Cursor steht, und wird benutzt, um das ursprüngliche Zeichen an die Position des Cursors zurückschreiben zu können.

96 $60 NEWROW (A)
Zeile, zu der ein DRAWTO- oder FILL-Befehl hinzielt. Bei den XL/XE-Geräten ist dieses Register nach 757 ($2F5) verlegt worden.

97,98 $61,$62 NEWCOL (A)
Spalte, zu der ein DRAWTO- oder FILL-Befehl geht. Bei den XL/XE-Geräten wurden diese Register nach 758,759 ($2F6,$2F7) verlegt.

96,97 $60,$61 FKDEF (X)
Zeiger (LSB/MSB) auf die acht Byte große Definitionstabelle der vier frei belegbaren Funktionstasten auf der Tastatur des 1200XL, die im XL/XE leicht nachgerüstet werden können, da sie in deren Betriebssystem auch vorhanden sind. Die Belegung sieht wie folgt aus:

Tastenkombination	Funktion
F1	Cursor hoch
F2	Cursor runter
F3	Cursor links
F4	Cursor rechts
SHIFT-F1	Home (Cursor nach oben links)
SHIFT-F2	Cursor nach unten links

Tastenkombination	Funktion
SHIFT-F3	Cursor an Zeilenanfang (physikalische Zeile)
SHIFT-F4	Cursor an Zeilenende (physikalische Zeile)

Die Control-Kombinationen können nicht umdefiniert werden. Sie stehen daher nicht in der 8 Byte großen FKDEF-Tabelle.

Tastenkombination	Funktion
CTRL-F1	Tastatur ab-/anschalten (nicht Konsole-Tasten)
CTRL-F2	DMA aus-/einschalten
CTRL-F3	Tastaturklick aus-/einschalten
CTRL-F4	Zeichensatz umschalten international/Standard

Die Kombination SHIFT-CTRL-Funktionstaste ist nicht belegt.

98 $62 PALNTS (X)
Flag für die Farbfernsehsystem-Version. 0 bedeutet NTSC (beispielsweise USA), 1 steht für PAL (Deutschland, England etc.).

99 $63 LOGCOL
Der ATARI kennt nicht nur physikalische Zeilen, sondern auch logische. Maximal drei physikalische Zeilen können eine logische Zeile darstellen. Diese logische Zeile wird vom Editor wie eine große physikalische Zeile behandelt. LOGCOL gibt nun die aktuelle Länge der logischen Zeile an. Der Wert in diesem Register kann also zwischen 0 und 119 liegen.

100,101 $64,$65 ADRESS
Diese beiden Register werden vom Betriebssystem bei verschiedenen Funktionen des Editors, wie zum Beispiel beim Verschieben des Bildschirms, Löschen o. ä., als Zero-Page-Zeiger benutzt.

102,103 $66,$67 MLTTMP
Diese Register werden vom Bildschirmtreiber temporär benutzt, insbesondere das erste Byte beim OPEN-Befehl.

104,105 $68,$69 SAVADR
Hier werden zeitweise die Daten unter dem Cursor abgelegt, außerdem Daten während der Verschiebung von Zeilen.

106 $6A RAMTOP
Dieses Register enthält die Gesamtseitenzahl des vorhandenen RAM-Speichers unterhalb Adresse $C000. RAMTOP*256 ergibt also die tat-

sächliche Zahl der Bytes. Dieser Wert wird während der Reset-Routine durch das Betriebssystem festgestellt.

Will man Speicherplatz z.B. für Player/Missile-Grafik oder Maschinensprachunterroutinen für BASIC reservieren, sollte man RAMTOP auf einen niedrigeren Wert setzen und einen GRAPHICS-Befehl ausführen, da sich der Bildschirmtreiber (Editor) in der Festlegung des Bildspeichers insofern an RAMTOP orientiert, als er den Bildspeicher direkt unter RAMTOP legt.

Vorsicht jedoch bei der Platzierung von wichtigen Daten direkt über RAMTOP! Der GRAPHICS-Befehl löscht beim ATARI 400 und 800 aufgrund eines Fehlers des Betriebssystems 64 Zeichen direkt oberhalb von RAMTOP, das Scrolling des Textfensters sogar bis zu 800 Zeichen.

107 $6B BUFCNT
Momentane Anzahl der Zeichen in der logischen Bildschirmzeile.

108,109 $6C,$6D BUFSTR
In diesem Register wird temporär die Adresse des augenblicklich editierten Zeichens abgespeichert.

110 $6E BITMSK
Hier werden temporär Daten (Bitmasken) zum Aufbau von Punktgraphiken abgespeichert.

111 $6F SHFAMT
Diese Speicherstelle wird dazu verwendet, in den Punktgrafikstufen einzelne Bytes nach links oder rechts zu verschieben, um die einzelnen Punkte auf dem Bildschirm ansprechen zu können. Vor dem Verschieben entspricht dieser Wert dem in DMASK (672; $2A0).

112,113 $70,$71 ROWAC
Während der PLOT- und DRAW-Funktion werden dieses und das folgende Register als Rechenregister benutzt. Dieses Register kontrolliert den Wert der Reihe des zu setzenden Punktes.

114,115 $72,$73 COLAC
Hier wird die Spaltennummer eines zu setzenden Bildpunktes abgelegt.

116,117 $74,$75 ENDPT
Hier wird die Adresse des zu setzenden Endpunktes einer Linie abgespeichert. Dieser Wert ist der größere von DELTAR und DELTAC.

118 $76 DELTAR
Hier wird die Differenz (der Delta-Wert) zwischen NEWROW (96; $60) und ROWCRS (84; $54) abgelegt.

119,120 $77,$78 DELTAC
Hier wird die Differenz (der Delta-Wert) zwischen NEWCOL (97,98; $61,$62) und COLCRS (85,86; $55,$56) abgelegt.

121 $79 ROWINC (A)
Um den Wert dieser Speicherstelle (+1 ($01) oder -1 ($FF)) wird die Cursorzeilennummer beim Ziehen einer Linie erhöht bzw. erniedrigt. Dieses und das nachstehende Register kontrollieren die Richtung des Linienziehens, sie sind also das Vorzeichen zu DELTAR und DELTAC. Dieses Register ist bei den XL/XE-Geräten nach 760 ($2F8) verlegt worden.

122 $7A COLINC (A)
Um den Wert dieser Speicherstelle (+1 oder -1) wird die Cursorspaltennummer beim Ziehen einer Linie erhöht bzw. erniedrigt. Dieses Register wurde bei den XL/XE-Geräten nach 761 ($2F9) verlegt.

121,122 $79,$7A KEYDEF (X)
Zeiger (LSB/MSB) auf die Tastendefinitionstabelle (→ Keyboard). Sie ist 192 Bytes lang und besteht aus drei jeweils 64 Bytes langen Abschnitten. (→ Kapitel 3).

123 $7B SWPFLG
Um die Kontrolle darüber zu behalten, ob die Cursordaten für das Text- und Grafikfenster ausgetauscht sind (dann ist dieses Register 255 ($FF)) oder nicht (dann ist dieses Register 0), wird diese Speicherstelle verwendet. Dieser Austausch von Cursordaten (zwischen den Speicherstellen 84 bis 95 ($54-$5F) und 656 bis 667 ($290-$29B) wird durch das Betriebssystem bei Verwendung von Grafikstufen mit Textfenster vorgenommen.

124 $7C HOLDCH
Vor der Abfrage der SHIFT- und CONTROL-Taste wird der Tastaturcode (siehe dazu auch CH (764; $2FC)) temporär hier abgespeichert.

125 $7D INSDAT
Hier wird temporär das Zeichen unter dem Cursor abgespeichert.

126,127 $7E,S7F COUNTR
Dies ist der eigentliche Zähler für die DRAW-Funktion. Es wird der jeweils größere Wert von DELTAR und DELTAC abgespeichert. Dieser Wert wird bis auf 0 dekrementiert. Beim Erreichen des Wertes 0 ist das Ziehen der Linie beendet.

Die Register 128-255 ($80-$FF) werden vom Betriebssystem nicht benutzt; sie sind für Anwenderprogramme frei. Lediglich 139, 140 ($8B, $8C) werden vom XL/XE-Betriebssystem temporär beim Kaltstart verwendet, um eine Prüfsumme über die letzten Bytes des Modul-ROMs zu berechnen. Die Benutzung von Seite 0 durch Programmiersprachen:

ATARI-BASIC:	128-202	($80-$CA)
	+210,211	($D2,$D3)
	+212-255	($D4-$FF) (Fließkommaroutinen)
ATARI-Assembler:	128-176	($80-$B0)
	+164-255	($A4-$FF) (nur beim "Debugging")
ACTION!:	128-202	($80-$CA)
	+206-212	($CE-$D4)
MAC/65:	128-255	($80-$FF)
BASIC XL:	128-203	($80-$CB)
	+212-255	($D4-$FF) (Fließkommaroutinen)

Folgende Speicherstellen werden von ATARI-BASIC benutzt:

128,129 $80,$81 LOMEM

Startadresse des BASIC-Programms, die bei der BASIC-Initialisierung aus MEMLO (743,744; $2E7,$2E8) übernommen wird. Die ersten 256 Bytes nach der durch LOMEM angegebenen Adresse sind der sogenannte Tokenausgabepuffer: Er wird dazu benötigt, BASIC-Befehle in die entsprechenden Token (Zahl, die im Programmspeicher diesen Befehl darstellt) umzuwandeln.

Daneben erfüllt dieser 256-Byte-Puffer noch weitere Aufgaben. LOMEM hat beim Laden und Speichern von BASIC-Programmen die folgende Bedeutung: Beim Speichern wird das BASIC-Programm in zwei Blöcken übertragen. Der erste Block (14 Bytes) enthält die Werte der sieben BASIC-Zeiger (LOMEM bis STARP; 128-141; $80-$8D), von denen jeweils der Inhalt von LOMEM subtrahiert ist. Der zweite Block besteht aus Variablennamentabelle, Variablenwertetabelle, dem Programm an sich und der Direkteingabezeile. Diese enthält den zuletzt im sogenannten "Direct"-Modus eingegebenen Befehl und hat immer die Zeilennummer 32768.

Beim Laden eines BASIC-Programms werden zunächst die sieben BASIC-Zeiger, zu denen MEMLO addiert wird, auf ihre Plätze zurückgeschrieben, anschließend wird das BASIC-Programm in den Speicher (256 Bytes oberhalb von MEMLO) geladen.

130,131 $82,$83 VNTP
Hier beginnt die Variablennamentabelle. Es sind bis zu 128 Variablen möglich, die im BASIC-Programm als Tokens von 128-255 ($80-$FF) dargestellt werden. Im Speicher ist jeweils das letzte Byte des Variablennamens invertiert (bei Zeichenketten das "$"-Zeichen, bei Variablenfeldern das "("-Zeichen). Die Namen der Variablen werden in der Reihenfolge abgespeichert, in der sie eingegeben werden.

132,133 $84,$85 VNTD
Diese Variable zeigt auf das erste Byte hinter der Variablennamentabelle.

134,135 $86,$87 VVTP
Zeiger auf die Variablenwertetabelle. Für jede Variable sind acht Bytes vorgesehen. Die Werte der Variablen werden wie folgt abgespeichert:

Einfache Variablen haben als erstes Byte eine 0, als zweites Byte die Variablennummer, die nächsten sechs Werte enthalten den Fließkommawert der Variablen.

Variablenfelder enthalten als erstes Byte den Wert 65, als zweites Byte wieder die Variablennummer, als drittes und viertes Byte den Offset zu STARP, als fünftes bis achtes Byte den ersten und zweiten Wert im DIM-Befehl.

Die Zeichenkettenvariablen haben als erstes Byte den Wert 129, als Zweites wieder die Variablennummer, als drittes und viertes Byte den Offset zu STARP, als fünftes und sechstes Byte die aktuelle Länge und als siebtes und achtes Byte die Länge, die im DIM-Befehl angegeben war.

Mit dem Begriff "Offset" ist die Zahl gemeint, die man zu dem in STARP angegebenen Wert hinzuaddieren muss, um die gewünschte Adresse zu erhalten.

136,137 $88,$89 STMTAB
Anfangsadresse des BASIC-Programms. Da den direkt eingegebenen Befehlen die Zeilennummer 32768 gegeben wird, steht am Schluss des Programms das zuletzt eingegebene Kommando, die sogenannte Direkteingabezeile. Am Anfang jeder Programmzeile befinden sich drei Bytes: Die ersten beiden enthalten die Zeilennummer in 2-Byte-Integer-Form, das dritte die Zahl der Bytes zur nächsten Programmzeile.

138,139 $8A,$8B STMCUR
Zeiger auf die zurzeit bearbeitete BASIC-Zeile.

140,141 $8C,$8D STARP
Diese Adresse gibt sowohl das Ende des BASIC-Programms an als auch den Beginn der Tabelle für die Werte der Variablenfelder und Zeichenketten. Jeder Wert aus den Variablenfeldern belegt sechs Bytes im Fließkommaformat, jedes Zeichen aus einer Zeichenkette jedoch nur ein Byte.

142,143 $8E,$8F RUNSTK
Adresse des Stapels für die GOSUB- und FOR-NEXT-Befehle. Jede Rücksprungadresse für einen GOSUB-Befehl belegt vier Bytes. Das erste Byte ist immer 0. Es dient dazu, den Eintrag auf dem Stapel als Rücksprungadresse eines GOSUB-Befehls zu markieren. Das zweite und dritte Byte (Low/High-Byte) ist die Zeilennummer, in der der GOSUB-Befehl erfolgte. Das vierte Byte gibt den Punkt (als Offset) innerhalb der Zeile an, in der der GOSUB-Befehl steht. Das ist notwendig, um die Abarbeitung der Zeile bei dem auf GOSUB folgenden Befehl fortsetzen zu können.

Der POP-Befehl manipuliert nur diese Argumente auf dem Stapel. Jedes FOR-NEXT-Argument auf dem Stapel sieht wie folgt aus: Die ersten sechs Bytes sind der Endwert der Schleife im Fließkommaformat. Die folgenden sechs Bytes geben die Schrittweite an. Das dreizehnte Byte ist die Variablennummer plus 128, das 14. und 15. Byte die Zeilennummer des FOR-Befehls, das 16. Byte der Offset in diese Zeile, um wie bei GOSUB innerhalb der Zeile das Programm fortsetzen zu können.

144,145 $90,$91 MEMTOP
Diese beiden Speicherstellen zeigen auf das höchste vom BASIC belegte Byte. Der Platz zwischen MEMTOP und dem Bildspeicher wird durch die FRE(0)-Funktion angegeben.

Die Speicherstellen 146-202 ($92-$CA) werden ebenfalls vom ATARI-BASIC belegt, nur einige davon können durch den Anwender sinnvoll abgefragt und interpretiert werden. Sie sind nachstehend aufgeführt.

186,187 $BA,$BB STOPLN
Nummer der Zeile, in der ein Fehler oder STOP-Befehl auftrat oder die BREAK-Taste gedrückt wurde.

195 $C3 ERRSAV
Nummer des Fehlers, der eine Programmunterbrechung oder einen TRAP ausgelöst hat.

201 $C9 PTABW
Definiert den Abstand der BASIC-Tabulator-Positionen für PRINT-Befehle. Bei einem Komma in einem PRINT-Befehl wird (mittels Ausgabe von Leerzeichen) zur nächsten BASIC-Tab-Position gesprungen. Der Standardwert

ist 10. Ist der linke Rand auf 2 gesetzt, liegen die BASIC-Tab-Positionen bei 2, 12, 22, 32, 42, 52 usw. Der Minimalwert ist 3. Diese BASIC-Tabs sind eine Funktion des BASIC-ROMs und haben daher nichts mit der TAB-Taste oder dem TAB-Zeichen des Bildschirm-Editors zu tun.

202 $CA LOADFLG
Wird dieses Flag auf 0 gesetzt, wird ein BASIC-Programm im Speicher gelöscht, sobald der BASIC-Interpreter in den "Immediate Mode" geht, also die Eingabe einer BASIC-Zeile bzw. -Kommandos erwartet. Das ist z.B. der Fall, wenn (bei laufendem BASIC-Programm) ein STOP oder END ausgeführt, die Break- oder Reset-Taste gedrückt oder vom DOS ins BASIC gegangen wird. Einige BASIC-Programme verwenden das als Listschutz.

203-209 $CB-$D1 FREE
Freie Speicherstellen.

210,211 $D2,$D3 FREE (BASIC)
Ursprünglich für ATARI BASIC oder andere Cartridges reserviert, sind diese Adressen offenbar unbenutzt geblieben.

212,213 $D4,$D5 BININT
Eine per USR-Funktion aufgerufene 6502-Routine kann hier den Returnwert für BASIC ablegen. Default ist 0. Die USR-Aufrufparameter liegen auf dem Stack: Das 1. Byte ist die Parameteranzahl, die folgenden Bytepaare bilden die Parameter (jeweils erst High-Byte, dann Low-Byte).

Es folgen nun ab 212 ($D4) die Fließkommaregister (→ Fließkommaroutinen).

212-217 $D4-$D9 FR0
Fließkommaregister 0 enthält sechs Bytes, die zusammen eine Zahl im Fließkommaformat darstellen. Fast alle Fließkommaroutinen benutzen dieses Register.

218-223 $DA-$DF FRE
Internes Fließkommaregister.

224-229 $E0-$E5 FR1
Fließkommaregister 1, siehe FR0.

230-235 $E6-$EB FR2
Fließkommaregister 2, siehe FR0.

Die Speicherstellen 236-241 ($EC-$F1) enthalten verschiedene Register für die Fließkommaroutinen.

236 $EC FRX
Freie Speicherstelle.

237 $ED EEXP
Dieses Register gibt den Wert des Exponenten an.

238 $EE NSIGN
Gibt das Vorzeichen der gerade bearbeiteten Fließkommazahl an.

239 $EF ESIGN
Dieses Register gibt das Vorzeichen des Exponenten an.

240 $F0 CHRFLG
Flag für das erste Zeichen.

241 $F1 DIGRT
Diese Speicherstelle enthält die Zahl der Stellen nach dem Komma.

242 $F2 CIX
Offset, der zu der Adresse des Bytes, das durch INBUFF adressiert wird, addiert werden muss. ASCII-Werte von dieser Adresse an aufwärts werden durch die Routine AFP (55296; $D800) in Fließkommaformat umgewandelt und in FR0 abgelegt.

243,244 $F3,$F4 INBUFF
Adresse einer Fließkommazahl als ASCII-Text. Wird von den Routinen zur Umwandlung von Fließkommazahlen in ASCII (FASC) und umgekehrt (AFP) benutzt: AFP erwartet in INBUFF einen Zeiger auf den ASCII-Text, FASC gibt einen solchen zurück, der innerhalb von LBPR1, LBPR2, LBUFF ($57E-$5FF) liegt.

Die Adressen 245-250 ($F5-$FA) stellen weitere temporäre Register dar, die von den Fließkommaroutinen benötigt werden.

245,246	$F5,$F6	ZTEMP1
247,248	$F7,$F8	ZTEMP2
249,250	$F9,$FA	ZTEMP3

251 $FB RADFLG
Der Standardwert 0 bedeutet, dass die im BASIC-ROM liegenden trigonometrischen Funktionen im Bogenmaß ausgeführt werden. Der Wert 6 in diesem Register hat zur Folge, dass die trigonometrischen Funktionen im Gradmaß ausgeführt werden.

252,253 $FC,$FD FLPTR
Diese Speicherstelle zeigt auf das zurzeit vom Benutzer belegte Fließkommaregister (→ Fließkommaroutinen).

254,255 $FE,$FF FPTR2
Zeiger auf das andere benutzte Fließkommaregister, das bei verschiedenen Rechenoperationen mitbenutzt wird.

1.2 Seite 1 (Page 1)

Seite 1 (256-511; $100-1FF) ist der Stapel für den Prozessor des ATARI.

1.3 Seite 2 (Page 2)

Die Seiten 2 bis 4 werden vom Betriebssystem für die unterschiedlichsten Aufgaben belegt. Hier stehen verschiedene Vektoren, Kontrollvariablen, Datenpuffer etc.

Die Speicherstellen 512-553 ($200–$229) sind die verschiedenen Interruptvektoren.

Grundsätzlich kennt der 6502-Prozessor des ATARI-Computers zwei verschiedene Arten von Interrupts: Der ANTIC-Chip kontrolliert die nichtmaskierbaren Interrupts (NMI: Non-Maskable-Interrupts) – sie sind daher sinnigerweise doch maskierbar. Die maskierbaren Interrupts (IRQ: Interrupt Request) werden durch den POKEY- und PIA-Chip überwacht.

Die folgenden Interrupts sind NMIs: Der Vertical-Blank-Interrupt, der Display-List-Interrupt und der RESET-Interrupt, ausgelöst durch die Reset-Taste bei den 400/800er Geräten. Bei den XL/XE-Geräten löst die Reset-Taste stattdessen einen echten CPU-Reset aus. Maskierbar sind der Tastatur-Interrupt, die Timer-Interrupts und Interrupts, die durch den seriellen Bus ausgelöst werden.

Für die NMIs und IRQs existieren zwei Betriebssystemroutinen, die die jeweilige Ursache des Interrupts feststellen und anschließend über den dazugehörigen Vektor die entsprechende Interrupt-Routine aufrufen. Wenn hier also davon die Rede ist, dass ein Interrupt "durch" einen Vektor springt, so ist damit gemeint, dass indirekt durch die ROM-Adressen 65530,65531 ($FFFA,$FFFB) im Falle der NMIs bzw. 65534,65535 ($FFFE, $FFFF) im Falle der IRQs gesprungen wird. Dabei wird jeweils eine Betriebssystem-Routine angesprungen, die für die Einzelfallbehandlung der verschiedenen Interrupts sorgt. Letztlich hat also jeder Interrupt einen zugehörigen Vektor. Detailliertere Informationen dazu sind auch bei NMIST (54287; $D40F) und IRQEN (53774; $D20E) zu finden.

512,513 $200,$201 VDSLST
Diese beiden Speicherstellen sind der Vektor für den sogenannten Display List-Interrupt (DLI). Dieser wird immer dann ausgelöst, wenn Bit 7 in einem Display-List-Befehl gesetzt ist. Der DLI wird zum Beispiel dazu benutzt, Farben oder Zeichensätze während des Bildaufbaus umzuschalten.

Das Verfahren zur Installation eines DLI sieht wie folgt aus: Nach Aufbau einer entsprechenden Display List und Erstellung einer Maschinensprachroutine, die sämtliche zu benutzende 6502-Register auf den Stapel retten und nach Ausführung der Routine wieder von diesem zurückholen muss, wird die Anfangsadresse dieser Routine in VDSLST abgelegt. Anschließend muss Bit 7 in NMIEN (54287; $D40E) gesetzt werden, um den Interrupt wirksam werden zu lassen. Am Ende der Maschinenspracheroutine muss in jedem Fall ein RTI-Befehl stehen. Um Farben im Interrupt flimmerfrei umzuschalten, sollte jeweils vor der Änderung des Farbregisters ein STA WSYNC (54282; $D40A) durchgeführt werden.

514,515 $202,$203 VPRCED
Der serielle Bus verfügt über eine sogenannte "Proceed Line", die einen Interrupt auslösen kann, der durch diesen Vektor adressiert wird. Im Augenblick wird dieser Proceed Line-Interrupt genauso wie der folgende Interrupt jedoch nicht benutzt, da es im Moment keine Peripheriegeräte gibt, die diese Interrupts erzeugen könnten.

516,517 $204,$205 VINTER
Ebenso wie der Proceed Line-Interrupt ist der serielle Interrupt, für den dieser Vektor vorhanden ist, bislang unbenutzt.

518,519 $206,$207 VBREAK
Der 6502-Prozessor verfügt über einen softwaremäßig erzeugbaren Interrupt. Dieses geschieht über den BRK-Befehl. Der durch den BRK-Befehl (nicht die BREAK-Taste!) erzeugte Interrupt hat als Vektor diese beiden Speicherstellen. Solch ein Interrupt ereignet sich, wenn ein Maschinensprachprogramm auf den Maschinensprachbefehl BRK ($00) trifft. Diese Interruptmöglichkeit kann zur Fehlersuche in Maschinenprogrammen (Debugging) verwendet werden.

520,521 $208,$209 VKEYBD
Vektor für den Tastaturinterrupt. Der Druck jeder Taste außer der RESET-, BREAK-, START-, OPTION-, SELECT-, SHIFT- und CONTROL-Taste löst einen Tastaturinterrupt aus. Das Betriebssystem initialisiert diesen Vektor auf eine ROM-Routine, die den Tastaturcode in das Register CH (764; $2FC) schreibt und u. a. für die Tastaturentprellung sorgt (→ Keyboard).

522,523 $20A,$20B VSERIN
Wenn der POKEY-Chip Daten über den seriellen Bus empfangen hat, löst dieser einen Interrupt aus, der über diesen Vektor die Daten vom seriellen Port in einen entsprechenden Puffer überträgt. Das DOS ändert diesen Vektor auf eine eigene Routine.

524,525 $20C,$20D VSEROR
Der Interrupt, für den dieser Vektor vorhanden ist, ist das Gegenstück zum vorhergehenden. Er sorgt nämlich dafür, dass, wenn der POKEY-Chip bereit zur weiteren Übertragung ist, Daten aus einem Puffer an den seriellen Port gegeben werden. Auch dieser Vektor wird durch das DOS verändert.

526,527 $20E,$20F VSEROC
Nach Übertragung aller Daten wird ein Interrupt ausgelöst, der XMTDON (58; $3A) setzt, um das Ende der Übertragung (einschließlich des Prüfsummenbytes) anzuzeigen. Auf diese Interruptroutine wird durch diesen Vektor gezeigt.

528,529 $210,$211 VTIMR1
Dieser Vektor wird dann durchsprungen, wenn ein Interrupt nach Herunterzählen von AUDF1 (53760; $D200) ausgelöst wird. Die AUDF-Zähler werden durch Schreiben eines beliebigen Wertes in STIMER (53769; $D209) gestartet. Siehe auch POKMSK (16; $10).

530,531 $212,$213 VTIMR2
Vektor für den Interrupt des zweiten POKEY-Zählers (AUDF2: 53762; $D202).

532,533 $214,$215 VTIMR4
Vektor für den Interrupt des vierten POKEY-Zählers (AUDF4: 53766; $D206). Dieser Interrupt ist aufgrund eines Fehlers im Betriebssystem der alten Geräte nur mit den XL/XE-Geräten über diesen Vektor nutzbar.

534,535 $216,$217 VIMIRQ
Falls ein maskierbarer Interrupt (IRQ) ausgelöst wird, muss das Betriebssystem zunächst feststellen, aus welchem Grund der Interrupt ausgelöst wurde. Dies geschieht durch einen indirekten Sprung durch dieses Register. Von der ROM-Routine, die durch dieses Register adressiert wird, wird dann in die einzelnen Interrupt-Routinen verzweigt. Siehe auch Register IRQEN (53774; $D20E).

Außer den Hardwarezählern (POKEY-Zähler) und der internen Uhr (RTCLOK: 18,19,20; $12,$13,$14) gibt es noch weitere Zähler, die wie RTCLOK softwaremäßig gesteuert werden. Diese System-Zähler mit ihren

zugehörigen Vektoren liegen in den Registern 536-558 ($218-$22E) und werden mit Ausnahme von CDTMV1 (536,537; $218,$219) während des zweiten Teils des System-Vertikal-Blanks verringert. Da der zweite Teil des System-Vertikal-Blanks während zeitkritischer I/O-Operationen nicht ausgeführt wird, werden in diesem Fall die System-Zähler außer CDTMV1 nicht weiter dekrementiert.

Welche Art von Zählern man für die jeweilige Anwendung benötigt, hängt von der Zeitspanne ab, die es zu überbrücken gilt. Bei Zeiträumen von weniger als 1/50 Sekunde muss man die POKEY-Zähler verwenden, bei Zeitspannen oberhalb dieses Wertes empfiehlt sich die Verwendung der System-Zähler.

536,537 $218,$219 CDTMV1
Dies ist der erste Systemzähler. Da er aus zwei Bytes besteht und jede 1/50 Sekunde um 1 erniedrigt wird, kann damit ein Zeitraum von maximal 21,845 Minuten überbrückt werden. Er wird als einziger Systemzähler im ersten Teil des System-Vertikal-Blank-Interrupts vermindert. Daher wird er während zeitkritischer I/O-Operationen nicht angehalten. Wenn dieser Zähler 0 erreicht, wird ein Sprung indirekt durch CDTMA1 (550,551; $226,$227) durchgeführt. Da das Betriebssystem diesen Zähler für verschiedene Aufgaben (zum Beispiel in der seriellen Ein- und Ausgaberoutine (SIO)) benötigt, wird dringend davon abgeraten, diesen Zähler für Benutzeranwendungen zu verwenden.

538,539 $21A,$21B CDTMV2
Diese zwei Bytes stellen den zweiten Systemzähler dar. Ähnlich wie bei CDTMV1 wird beim Erreichen von 0 indirekt durch CDTMA2 (552,553; $228,$229) gesprungen. Dieser Zähler steht für den Benutzer frei zur Verfügung, wird aber aufgrund des oben geschilderten Sachverhaltes nicht in jedem Fall dekrementiert.

540,541 $21C,$21D CDTMV3
Der dritte Systemzähler wird vom Betriebssystem beim Öffnen eines Kassettenfiles benötigt, ansonsten ist er frei verwendbar. Im Gegensatz zu den Zählern 1 und 2 wird hier, ebenso wie bei den weiteren beiden Zählern, nach dem Erreichen von 0 nur ein Flag (CDTMF3: 554,$22A) gesetzt.

542,543 $21E,$21F CDTMV4
Dies ist der vierte Systemzähler. Das zugehörige Flag zur Anzeige des Erreichens von 0 ist CDTMF4 (556,$22C).

544,545 $220,$221 CDTMV5
Der fünfte Systemzähler mit zugehörigem Flag CDTMF5 (558,$22E).

546,547 $222,$223 VVBLKI

Dieser Vektor wird nach dem Ausführen des ersten Teiles des System-Vertikal-Blanks (sog. "Immediate VBlank") durchsprungen. Er ist auf den ersten Teil des System-Vertikal-Blanks initialisiert, dessen Einsprungadresse SYSVBV (58463; $E45F) ist. Der Benutzer kann nun diesen Vektor auf eine eigene Routine zeigen lassen. Die Initialisierung dieser Routine erfolgt durch einen Einsprung in die SETVBV-Routine (58460; $E45C), wobei der Akkumulator den Wert 6, das X-Register das Hi-Byte der Adresse und das Y-Register das Low-Byte der Adresse der Routine enthalten muss. Am Ende der Routine muss ein JMP SYSVBV stehen. Diese neu installierte Routine wird dann, ebenso wie der erste Teil des System-Vertikal-Blanks, jede 1/50 Sekunde ausgeführt. Sie kann etwa 4500 Maschinenzyklen umfassen, ohne dass Probleme auftreten, sofern keine SIO-Routinen verwendet werden.

548,549 $224,$225 VVBLKD

Diese zwei Bytes stellen den Vektor für den sogenannten "Deferred VBlank" dar. Das ist der Teil des Vertikal-Blank-Interrupts, der nach dem Ausführen des System-Vertikal-Blanks durchsprungen wird. Dieses Register ist auf XITVBV (58466; $E462) initialisiert, kann aber durch Benutzung der Routine SETVBV (58460; $E45C) auf eine eigene Routine verändert werden. Der Akkumulator muss in diesem Fall den Wert 7 enthalten. Am Schluss dieser neuen Routine muss dann ein JMP XITVBV stehen. Für Benutzeranwendungen stehen etwa 24.000 Maschinenzyklen zur Verfügung. Der Deferred VBlank wird dann nicht ausgeführt, wenn CRITIC (66; $42) auf 1 gesetzt ist, d.h., wenn im Augenblick eine zeitkritische I/O-Operation stattfindet.

550,551 $226,$227 CDTMA1

Wenn der erste Systemzähler den Wert 0 erreicht, wird ein Sprung durch diese Adresse durchgeführt. Am Ende der angesprungenen Routine sollte ein RTS-Befehl stehen. Siehe auch CDTMV1 (536,537; $218,$219).

552,553 $228,$229 CDTMA2

Sprungadresse für den zweiten Systemzähler (CDTMV2: 538,539; $21A, $21B).

554 $22A CDTMF3

Diese Speicherstelle wird dann auf einen Wert ungleich 0 gesetzt, wenn der dritte Systemzähler (CDTMV3: 540,541; $21C,$21D) auf 0 dekrementiert worden ist.

555 $22B SRTIMR

Dies ist der Zähler, der für die Tastaturwiederholfunktion zuständig ist. Jedes Mal, wenn eine Taste gedrückt wird, wird bei den ATARI-400/800-

Geräten in dieses Register der Wert 48 ($30) hineingeschrieben, bei den XL/XE-Geräten der Wert aus KRPDEL (729; $2D9). Während des zweiten Teils des System-Vertikal-Blanks wird dieses Register so lange dekrementiert, bis es 0 erreicht. Ist dann die Taste immer noch gedrückt, wird bei den ATARI 400/800-Geräten jede 1/10 Sekunde eine Wiederholung dieses Tastaturwertes, der in CH (764; $2FC) zu finden ist, durchgeführt. Bei den XL/XE-Geräten richtet sich die Wiederholrate nach dem Wert in KEYREP (730; $2DA). SRTIMR kontrolliert also die Ansprechzeit der Wiederholfunktion.

556 $22C CDTMF4

Wenn der vierte Systemzähler (542,543; $21E,$21F) auf 0 dekrementiert ist, wird dieses Flag gesetzt.

557 $22D INTEMP

Dieses Register wird nur von der SETVBV-Routine (58460; $E45C) verwendet, die zum Setzen der Systemzähler und der Vertikal-Blank-Interrupt-Vektoren dient.

558 $22E CDTMF5

Flag für den fünften Systemzähler CDTMV5 (544,545; $220,$221).

559 $22F SDMCTL

Diese Speicherstelle ist das Schattenregister zu DMACTL (54272; $D400) und kontrolliert den direkten Speicherzugriff ("Direct Memory Access", DMA) des ANTIC auf den Speicher des Computers. Während des direkten Speicherzugriffs wird die 6502-CPU angehalten. Um Programme zu beschleunigen (ca. 30%), kann also der DMA des ANTIC abgeschaltet werden. Die einzelnen Bits dieses Registers kontrollieren das Aussehen des Bildschirms wie folgt:

Bit	Dez	Hex	Funktion
5	32	$20	DMA für die Display List ein/aus
4	16	$10	Einzeilige P/M-Auflösung ein/aus
3	8	$8	DMA für Player ein/aus
2	4	$4	DMA für Missiles ein/aus
0,1	3	$3	Breites Anzeigefeld (48 Zeichen)
0,1	2	$2	Normales Anzeigefeld (40 Zeichen)
0,1	1	$1	Schmales Anzeigefeld (32 Zeichen)
0,1	0	$0	Kein Anzeigefeld

Falls die einzeilige P/M-Auflösung (Bit 4) nicht gewünscht wird, so werden die Player und Missiles in zweizeiliger Auflösung dargestellt. Die Angabe der Zeichenbreite für das Anzeigefeld bezieht sich natürlich auf die Grafikstufe 0. Diese Zahlen der Zeichen entsprechen 192, 160 oder 128 Farbpunkten ("Color clocks"). Insgesamt stellt der ANTIC 228 Farbpunkte (einschließlich des Randes) dar, sichtbar sind jedoch nur je nach Fernseher/Monitor ca. 174 davon. Beim Einschalten des breiten Anzeigefelds sind deshalb auch nicht alle 48 Zeichen sichtbar. Die Standardeinstellung für dieses Register ist 34 ($22). Dabei sind die Bits 1 und 5 gesetzt. Genauere Informationen über die Player/Missile-Grafik sind in dem entsprechenden Abschnitt in Kapitel 2 zu finden.

560,561 $230,$231 SDLSTL, SDLSTH

An der durch diese beiden Speicherstellen angegebenen Adresse beginnt die Display List. Sie ist sozusagen das Programm für den ANTIC. Dieses Programm enthält die Informationen über die Adresse der Bildschirmdaten und die Art und Weise, in der sie darzustellen sind. SDLSTL ist das Schattenregister zu DLISTL/H (54274,54275; $D402,$D403). MEMTOP (741,742; $2E5,$2E6) wird vom Betriebssystem direkt unterhalb der Display List gesetzt.

Genauere Informationen über Display Lists und Display List Interrupts finden sich im Abschnitt über den ANTIC.

562 $232 SSKCTL

Dieses Register kontrolliert die serielle Schnittstelle des Computers. Es ist das Schattenregister zu SKCTL (53775; $D20F). Die einzelnen Bits dieser Speicherstelle haben die folgende Bedeutung:

Bit	Dez	Hex	Funktion
7	128	$80	Ein Setzen des Bits führt zur Unterbrechung der Übertragung, es wird 0 ausgegeben.
6	64	$40	Diese drei Bits kontrollieren die Art
5	32	$20	der Ein- und Ausgabe über die serielle
4	16	$10	Schnittstelle.
3	8	$8	Der POKEY-Zwei-Kanal-Modus wird eingeschaltet.
2	4	$4	Die Drehregler werden schneller gelesen
1	2	$2	Tastatur ein/aus
0	1	$1	Tastaturentprellung ein/aus

Die Bits 4 bis 6 kontrollieren die Art der Ein- und Ausgabe über den seriellen Port. Da eine andere als die durch das Betriebssystem (→ SIO) unterstützte Verwendung der seriellen Schnittstelle nicht bekannt und auch sinnvoll ist, sei hier auf das Hardwaremanual verwiesen. Daneben gibt es keine externen Geräte, die zwar über den seriellen Bus arbeiten, jedoch nicht über die SIO angesprochen werden können.

Beim Setzen von Bit 3 werden die Daten über den seriellen Port auf den Kassettenrecorder nicht als logisch wahr/falsch, sondern als zwei Töne übertragen. Dabei gibt Tonkanal 1 die logisch wahren, Tonkanal 2 die logisch falschen Informationen.

Falls Bit 2 gesetzt ist, werden die Drehregler nicht während eines kompletten Bildschirmaufbaus, sondern, wesentlich schneller, innerhalb von zwei Bildschirmzeilen ausgelesen.

Das Register ist auf 3 ($3, Bits 0 und 1 gesetzt) initialisiert.

563 $233 LCOUNT (X)
Diese Speicherstelle dient als Hilfsregister für die Gerätetreiberladeroutine (Zähler für die Länge der Einträge).

564 $234 LPENH
Der Wert in dieser Speicherstelle liegt zwischen 0 und 227 ($E3) und gibt die horizontale Spaltenposition des Lightpens an. Vom linken bis zum rechten Bildschirmrand betragen die Werte 92-227, 0-23. Schattenregister zu PENH (54284; $D40C).

565 $235 LPENV
Dieses Register gibt die vertikale Position für den Lightpen an. Es enthält die Bildschirmzeilennummer geteilt durch 2 und kann daher Werte von 4 (oben) bis 123 (unten) annehmen. Beim normalen Bildschirm in GRAPHICS 0 liegen die Werte von 16 (oben) bis 111 (unten).

566,567 $236,$237 BRKKY (X)
Bei den XL/XE-Geräten gibt dieser Vektor die Adresse an, zu der beim Drücken der BREAK-Taste gesprungen wird.

568,569 $238,$239 VPIRQ(X)
Diese Speicherstelle ist der IRQ-Vektor für den parallelen Bus.

Die Speicherstellen 570-573 ($23A-$23D) sind für die seriellen I/O-Operationen als Kommandopuffer reserviert. Diese Werte sollten vom Benutzer nicht verändert werden, da sie intern vom Betriebssystem benutzt werden.

570 $23A CDEVIC
Dieser Wert entspricht dem Inhalt von DDEVIC (768; $300) plus DUNIT (769; $301) minus 1.

571 $23B CCOMND
Das SIO-Kommandobyte wird aus DCOMND (770; $302) kopiert.

572,573 $23C,$23D CAUX1,2
Dies sind die Kommandohilfsbytes 1 und 2, die aus DAUX1,2 (778,779; $30A,$30B) gelesen werden. Bei der Benutzung der Diskettenstation geben sie zum Beispiel den anzusprechenden Sektor an.

574 $23E TEMP
Diese Speicherstelle wird von der SIO als temporäres Register verwendet.

575 $23F ERRFLG
Jeder Fehler des seriellen Busses außer Timeout setzt dieses Flag.

Die ersten vier Bytes aus dem ersten Sektor einer Bootdiskette oder -kassette werden während des Bootvorgangs in die folgenden vier Speicherstellen übertragen:

576 $240 DFLAGS
Das erste Byte jedes Bootsektors oder -records ist unbenutzt.

577 $241 DBSECT
Hier ist die Anzahl der zu ladenden Sektoren bzw. Records zu finden.
(1-255; $01-$FF → 0; $00 bedeutet 256; $100)

578,579 $242,$243 BOOTAD
Von dieser Adresse an aufwärts werden die durch DBSECT angegebenen Sektoren bzw. Records abgelegt.

580 $244 COLDST
Jede Zahl ungleich 0 in dieser Speicherstelle bewirkt, dass beim Drücken von RESET ein Kaltstart (Sprung durch COLDSV (58487; $E477) ausgeführt wird. Beim Wert 0 (dieser Wert ist normal) geschieht ein Warmstart, bei dem durch WARMSV (58484; $E474) gesprungen wird.

581 $245 RECLEN (X)
Das Byte gibt die Länge der Einträge für die Gerätetreiberladeroutine an.

582 $246 DSKTIM
Die DSKINV-Routine (58451; $E453) legt beim STATUS-Befehl hier den maximalen Timeout-Wert (beim Formatieren) der Diskettenlaufwerke ab.

583-622 $247-$26E LINBUF (A)
Dieser Puffer (40 ($28) Zeichen lang) wird vom Bildschirmtreiber benötigt, um hier temporär Daten während des Verschiebens des Bildschirminhaltes zu speichern. Auf diesen Puffer wird durch ADRESS (100,101; $64,$65) gezeigt. Bei den XL/XE-Geräten ist dieser Puffer entfallen.

583 $247 PDVMSK (X)
Jedes der acht Bits steht für ein an den parallelen Bus angeschlossenes Gerät, wobei ein gesetztes Bit signalisiert, dass das Gerät angeschlossen ist. 8 ist somit auch die Höchstzahl von Geräten, die gleichzeitig an den parallelen Bus angeschlossen werden können.

584 $248 SHPDVS (X)
Schattenregister zu PDVS (53759; $D1FF). Es entspricht im Aufbau PDVMSK, d.h., dass jedes Bit ein Peripheriegerät symbolisiert. Über dieses Register wird festgelegt, welches der am parallelen Bus angeschlossenen Geräte angesprochen werden soll.

585 $249 PDIMSK (X)
IRQ-Auswahlregister für den parallelen Bus.

586,587 $24A,$24B RELADR (X)
Zwischenregister für die Gerätetreiberladeroutine.

588 $24C PPTMPA (X)
Zwischenregister für die Treiberroutine für den parallelen Bus.

589 $24D PPTMPX (X)
Zwischenregister für die Treiberroutine für den parallelen Bus.

590-618 $24E-$26A ... (X)
Dieser Speicherbereich ist reserviert.

619 $26B CHSALT (X)
Zeiger auf den verwendeten Zeichensatz. Dieses Register wird von der Tastatur-Interrupt-Routine dazu verwendet, zwischen dem normalen und dem internationalen Zeichensatz umzuschalten, was allerdings nur beim ATARI 1200XL durch Drücken von CONTROL-F4 möglich ist.

620 $26C VSFLAG (X)
Interner Zähler für die vertikale Feinverschiebung, die vom Editor im sogenannten Fine-Scroll-Modus verwendet wird. Siehe → FINE (622; $26E).

621 $26D KEYDIS (X)
Ein- und Ausschalter für die Tastatur. 0 bedeutet hier, dass die Tastatur nicht gesperrt ist. Beim 1200XL kann die Tastatur über CONTROL-F1 ein- und ausgeschaltet werden.

622 $26E FINE (X)
Flag für den Fine-Scroll-Modus des Editors. Wenn beim Öffnen des Bildschirms (z. B. bei GRAPHICS 0) in diesem Register 255 steht, wird der Fine-Scroll-Modus eingeschaltet.

623 $26F GPRIOR
Dieses Register, Schattenregister zu PRIOR (53275; $D01B), kontrolliert den GTIA-Chip, der die Player/Missiles verwaltet, die Signale des ANTIC-Chips verarbeitet und diese Daten zum Fernseher/Monitor schickt. Es bestehen nun über dieses Register einige Manipulationsmöglichkeiten der Player/Missiles, die durch folgende Bits kontrolliert werden:

Bit	Dez	Hex	Funktion
7,6	192	$C0	GTIA-Modus 3: 16 Farben, eine Helligkeit (wie GRAPHICS 11). Die Auswahl der Helligkeitsstufe erfolgt über das Register COLOR4 (712; $2C8).
7,6	128	$80	GTIA-Modus 2: 9 Farben + Helligkeiten (wie GRAPHICS 10). Die Auswahl der Farben und Helligkeiten erfolgt über die acht Register PCOLR1-COLOR4 (705-712; $2C1-$2C8). Die Hintergrundfarbe wird durch das Register PCOLR0 (704; $2C0) festgelegt.
7,6	64	$40	GTIA-Modus 1: eine Farbe in 16 Helligkeiten (wie GRAPHICS 9). Die Auswahl der Farbe erfolgt über das Register COLOR4 (712; $2C8).
5	32	$20	Die Überlappung bestimmter Player ergibt eine dritte Farbe.
4	16	$10	Die vier Missiles werden zum fünften Player zusammengeschaltet.
			Verschiedene Rangfolgen bei Überlappungen der Anzeigefeld-Farben und Player sind wie folgt möglich:
3	8	$8	Anzeigefeld 0-1, Player 0-3, Anzeigefeld 2-3, Hintergrund
2	4	$4	Anzeigefeld 0-3, Player 0-3, Hintergrund
1	2	$2	Player 0-1, Anzeigefeld 0-3, Player 2-3, Hintergrund
0	1	$1	Player 0-3, Anzeigefeld 0-3, Hintergrund

Die Bits 6 und 7 dieses Registers wählen die verschiedenen GTIA-Stufen an. Die Pixel haben bei Einschaltung eines GTIA-Modus vierfache Breite; das heißt, dass horizontal nur noch 80 Punkte möglich sind, die aber, wie oben beschrieben, entweder eine von 16 Farben, eine von 16 Helligkeitsstufen oder einen Farbwert von neun möglichen annehmen können. Zur Funktion des Bits 5: Normalerweise ergibt die Überlappung von Playern an der betreffenden Stelle ein schwarzes Feld. Ist jedoch Bit 5 von GPRIOR gesetzt, so erzeugen die Überlappungen von Player 0/1 bzw. Player 2/3 eine dritte Farbe, die sich aus der logischen Oderierung der beiden betreffenden Farbregister ergibt.

Falls Bit 4 gesetzt ist, nehmen sämtliche Missiles die Farbe des Registers COLOR3 (711,$2C7) an. Die Positionen müssen jedoch weiterhin getrennt gesetzt werden.

Die Speicherstellen 624-647 ($270-$287) enthalten die Werte für die Joysticks, Paddles, Maltafel, Maus und andere Eingabegeräte, die an den Joystickport angeschlossen werden können.

624 $270 PADDL0
625 $271 PADDL1
626 $272 PADDL2
627 $273 PADDL3
628 $274 PADDL4
629 $275 PADDL5
630 $276 PADDL6
631 $277 PADDL7

Die Paddlewerte liegen zwischen 0 und 228 ($E4), sie werden bei Drehung im Uhrzeigersinn kleiner. Da bei den XL/XE-Geräten "nur" noch vier Paddles angeschlossen werden können, sind in diesem Fall nur die ersten vier Speicherstellen relevant. PADDL4-7 sind in diesem Fall Kopien von PADDL0-3. Die acht Speicherstellen sind die Schattenregister für POT0-7 (53760-53767; $D200-$D207).

632 $278 STICK0
633 $279 STICK1
634 $27A STICK2
635 $27B STICK3

Diese vier Speicherstellen enthalten die entsprechenden Joystickwerte, sie sind außerdem die Schattenregister zu PORTA/B (54016,54017; $D300,$D301). Bei den XL/XE-Geräten sind wiederum nur die ersten beiden Speicherstellen interessant. STICK2 und STICK3 sind dann Kopien von STICK0 bzw. STICK1.

636 $27C PTRIG0
637 $27D PTRIG1
638 $27E PTRIG2
639 $27F PTRIG3
640 $280 PTRIG4
641 $281 PTRIG5
642 $282 PTRIG6
643 $283 PTRIG7
Hier werden die Zustände der Paddleknöpfe abgespeichert: 0 bedeutet, dass der Knopf gedrückt ist, beim Wert 1 ist der Knopf nicht gedrückt. PTRIG0-3 sind die Schattenregister zu PORTA (54016; $D300), PTRIG4-7 zu PORTB (54017; $D301). Da bei den XL/XE-Geräten PORTB zur Speicherverwaltung verwendet wird, sind nur die ersten vier Speicherstellen wichtig. PTRIG4-7 sind dann Kopien von PTRIG0-3.

644 $284 STRIG0
645 $285 STRIG1
646 $286 STRIG2
647 $287 STRIG3
Diese Register, Schattenregister zu TRIG0-3 (53264-53267; $D010-$D013), geben auf dieselbe Art und Weise wie PTRIG0-3 den Zustand der Joystickknöpfe wieder. Beim Wert 0 ist der Knopf gedrückt, beim Wert 1 ist er losgelassen. Da bei den XL/XE-Geräten nur zwei Joysticks angeschlossen werden können, sind nur die ersten beiden Speicherstellen relevant. STRIG2 und STRIG3 sind beim XL/XE Kopien der ersten beiden.

648 $288 CSTAT (A)
Diese Speicherstelle ist ein Statusregister für den Kassettenrecorder, das bei den XL/XE-Geräten nicht mehr vorhanden ist.

648 $288 HIBYTE (X)
Zwischenregister für die Gerätetreiberladeroutine.

649 $289 WMODE
Betriebsart für den Kassettenrecorder. 128 ($80) bedeutet, dass geschrieben wird, 0 bedeutet, dass gelesen wird.

650 $28A BLIM
Der Wert dieser Speicherstelle kann zwischen 0 und 128 ($80) liegen; er gibt die aktuelle Größe des Kassettenpuffers an. Auf das zurzeit geschriebene oder gelesene Byte wird durch BPTR (61; $3D) gezeigt.

651 $28B IMASK (X)
Dieses Byte wird nicht verwendet, ist aber reserviert.

652,653 $28C,$28D JVECK (X)
Hier legt die System-IRQ-Routine die Startadresse der aufzurufenden IRQ-Routine ab - wie z.B. die Tastaturinterrupt-Routine.

654,655 $28E,$28F NEWADR (X)
Adressvariable für die Gerätetreiberladeroutine.

Die Speicherstellen 656-703 ($290-$2BF) sind vom Bildschirmeditor belegt. Bei der Benutzung von Textfenstern (GRAPHICS 1-8, 12-15) wird der Grafikteil des Bildschirms vom Bildschirmtreiber (S:) verwaltet, der Textteil vom Editor (E:).

656 $290 TXTROW
Dies ist die Reihe des Cursors im Textfenster. Der Wert kann zwischen 0 und 3 liegen, da das Textfenster nur 4 Zeilen hoch ist.

657,658 $291,$292 TXTCOL
Spalte des Cursors im Textfenster. Da das Textfenster grundsätzlich in der Grafikstufe 0 ist, kann hierbei der Wert zwischen 0 und 39 ($27) rangieren, Speicherstelle 658 ist deshalb immer 0. Da der POSITION-Befehl von BASIC sich immer auf den Grafikteil des Bildschirms bezieht, muss bei einer Veränderung der Cursorposition im Textfenster direkt in TXTCOL und TXTROW geschrieben werden.

659 $293 TINDEX
Die Grafikstufe des Textfensters ist hier abgespeichert. Dieses Register ist mit DINDEX (87; $57) vergleichbar. Es ist daher meistens 0.

660,661 $294,$295 TXTMSC
Hier findet sich die Adresse der linken oberen Ecke des Textfensters. Siehe auch SAVMSC (88,89; $58,$59).

662-667 $296-$29B TXTOLD
Diese sechs Speicherstellen dienen als Zwischenspeicher für TXTROW, TXTCOL, TINDEX und TXTMSC. Sie sind in ihrer Funktion vergleichbar mit OLDROW-OLDADR (90-95; $5A-$5F).

668 $29C TMPX1 (A)
Temporäres Register für den Bildschirmtreiber.

668 $29C CRETRY (X)
Maximale Anzahl der Wiederholversuche, einen Befehl an eins der verschiedenen externen Geräte zu geben. Dieses Register wird auf 13 ($D) initialisiert.

669 $29D HOLD3
670 $29E SUBTMP
671 $29F HOLD2
Drei weitere temporäre Register für den Bildschirmtreiber.

672 $2A0 DMASK
In dieser Speicherstelle ist jedes Bit gesetzt, das, je nach Grafikstufe, dem Pixel ("Picture cell", Bildschirmelement) entspricht, das gerade vom Bildschirmtreiber verarbeitet wird. Folgende kleine Tabelle macht dies in Abhängigkeit zur Grafikstufe deutlich:

Grafikstufen 4,6,8: 10000000
(8 Pixel pro Byte) 01000000
00100000
00010000
00001000
00000100
00000010
00000001

Grafikstufen 3,5,7: 11000000
(4 Pixel pro Byte) 00110000
00001100
00000011

Grafikstufen 9 ,10 ,11: 11110000
(zwei Pixel pro Byte) 00001111

Grafikstufen 0,1,2: 11111111
(ein Pixel pro Byte)

Jedes Pixel ist ein vom Bildschirm-Handler zu verarbeitender Punkt, der je nach Grafikstufe verschiedene Größen annehmen kann. Die folgende Tabelle zeigt den Zusammenhang von Grafikstufe und Pixelgröße:

	Textmodi				**Grafikmodi**					
Grafikstufe	0	1	2		3	4	5	6	7	8
Bildschirmzeile pro Zeichen	8	8	16		8	4	4	2	2	1
Bits pro Pixel	1	1	1		2	1	2	1	2	1
Farbpunkte pro Pixel	0,5	1	1		4	2	2	1	1	0,5
Zeichen pro Zeile	40	20	20		-	-	-	-	-	-
Pixel pro Zeile	-	-	-		40	80	80	160	160	320

673 $2A1 TMPLBT
Temporäres Register, das in Verbindung mit DMASK benutzt wird.

674 $2A2 ESCFLG
Dieses Register enthält 128 ($80), wenn die zuletzt gedrückte Taste ESCAPE war. Der Normalzustand nach Ausgabe des nächsten Zeichens ist 0. Um Steuerzeichen wie CLEAR, INSERT, DELETE usw. auf dem Bildschirm darzustellen, kann man einfach einen beliebigen Wert ungleich von 0 in dieses Register schreiben. Siehe auch DSPFLG (766, $2FE).

675-689 $2A3-$2B1 TABMAP
Für jede der 120 möglichen Stopppositionen innerhalb einer logischen Zeile nach dem Drücken der TAB-Taste ist in diesen 15 Bytes ein Bit reserviert. Ist das Bit gesetzt, wird in dieser Spalte angehalten. Da jede logische Zeile bis zu drei physikalische Bildschirmzeilen umfassen kann, sind 15 Bytes notwendig. Das höchstwertige Bit des ersten Bytes entspricht der ersten Spalte, das niederwertigste der achten Spalte usw. Die Tabelle wird nach dem Drücken von RESET oder Wechsel der Grafikstufe auf den Ursprungszustand zurückgesetzt. TABMAP kann in Grafikstufe 0 und in Textfenstern verwendet werden.

690-693 $2B2-$2B5 LOGMAP
In den ersten drei Bytes von LOGMAP ist jedes Bit (24 insgesamt) gesetzt, das den Beginn einer logischen Zeile enthält. Die Reihenfolge ist ähnlich wie bei TABMAP: Das höchstwertige Bit des ersten Bytes entspricht der ersten Zeile, das niederwertigste der achten usw. Das vierte Byte wird nicht verwendet, ist aber reserviert.

694 $2B6 INVFLG
Ist der Wert dieses Registers 0, werden alle Zeichen normal ausgegeben, ist der Wert 128 ($80), so werden alle Zeichen invers auf den Bildschirm gebracht. Das Ändern dieses Registers entspricht dem Drücken der ATARI- bzw. Inverse-ON/OFF-Taste auf Ihrer Tastatur.

695 $2B7 FILFLG
Steht in dieser Speicherstelle ein Wert ungleich 0, so ist die augenblickliche Grafikoperation des FILL-Kommando, ansonsten ist der auszuführende Befehl ein DRAW-Befehl.

696 $2B8 TMPROW
697,698 $2B9,$2BA TMPCOL
Dies sind temporär verwendete Speicher für OLDROW und OLDCOL (84-86; $54-$56).

699 $2BB SCRFLG

Diese Flagge wird gesetzt, sobald ein Scrolling des Bildschirms erfolgt. Vorsicht! Das Scrolling kann beim ATARI 400/800 bis zu 800 Zeichen oberhalb von RAMTOP (106; $6A) löschen. Bei den XL/XE-Geräten ist dieser Fehler korrigiert.

700 $2BC HOLD4 (A)
701 $2BD HOLD5 (A)

Zwei weitere Speicherstellen, die nur während der Ausführung des DRAW-Kommandos verwendet werden.

701 $2BD DRETRY (X)

Höchstzahl der Wiederholversuche (normaler Wert ist 1) des Peripheriegerätes, einen Befehl auszuführen.

702 $2BE SHFLOK

Diese Speicherstelle enthält 0, wenn auf Kleinschrift geschaltet ist, 64 ($40), wenn Großschrift eingeschaltet und 128 ($80), wenn auf die Control-Zeichen umgestellt ist. Diese Werte stehen nach dem Drücken der CAPS/LOWR-Taste allein oder zusammen mit SHIFT bzw. CONTROL. Dieses Register kann vom Benutzer verändert werden, um das Drücken der übrigen Tasten zu simulieren; Versuche, andere als die obengenannten Werte in diese Speicherstelle zu schreiben, können jedoch zum Absturz des Systems führen!

703 $2BF BOTSCR

Die Zahl der Zeilen in Grafikstufe 0 auf dem Bildschirm wird hier abgespeichert. Relevant sind jedoch nur die Zahlen 24, 4 oder 0. 24 entspricht dabei der normalen Grafikstufe 0, 4 einem Textfenster, 0 keinen durch den Editor veränderbaren Textzeilen. Andere Werte werden ignoriert. Die einzig sinnvolle Anwendung ist das Einschreiben des Wertes 4 in der Grafikstufe 0, um ein Textfenster zu erzeugen.

Die folgenden acht Speicherstellen sind die Farbregister für Player/Missiles und Anzeigefelder, gleichzeitig sind sie Schattenregister für COLPM0-3 (53266-53269; $D012-$D015), COLPF0-3 (53270-53273; $D016-$D019) und COLBK (53274; $D01A). Eine Farbtabelle findet sich im Tabellenteil. Der Farbwert errechnet sich generell wie folgt:

Farbwert = Helligkeit + 16 * Farbe.

Helligkeiten und Farben können zwischen 0 und 15 ($F) liegen, bei Helligkeiten wird jedoch, außer in den Grafikstufen 9 bis 11 (GTIA-Modi 1-3), das niederwertigste Bit ignoriert.

704 $2C0 PCOLR0
Das Register enthält die Farbe von Player und Missile 0. Im GTIA-Modus 2 (GRAPHICS 10) enthält es die Farbe für den Hintergrund, die normalerweise in COLOR4 (712; $2C8) zu finden ist.

705 $2C1 PCOLR1
706 $2C2 PCOLR2
707 $2C3 PCOLR3
Farben der Player und Missiles 1, 2 und 3.

708 $2C4 COLOR0
Farbregister für Anzeigefeld 0. In GRAPHICS 1 und 2 die Farbe der Großbuchstaben.

709 $2C5 COLOR1
Farbe für Anzeigefeld 1. In den Grafikstufen 1 und 2 wird in COLOR1 die Farbe für die Kleinbuchstaben abgelegt. In GRAPHICS 0 und 8 steht in dieser Speicherstelle der Helligkeitswert für die Buchstaben bzw. Punkte.

710 $2C6 COLOR2
Farbe für Anzeigefeld 2. Hier wird in GRAPHICS 1 und 2 die Farbe für die inversen Großbuchstaben abgelegt, in Grafikstufe 0 und 8 die Hintergrundfarbe der Buchstaben bzw. Punkte.

711 $2C7 COLOR3
Farbregister für Anzeigefeld 3. In Grafikstufe 1 und 2 steht hier der Farbwert für die inversen Kleinbuchstaben.

712 $2C8 COLOR4
In allen Grafikstufen außer GRAPHICS 10 findet sich hier die Hintergrundfarbe (Rahmenfarbe). In GRAPHICS 10 wird COLOR4 zu einem normalen Farbregister, während hingegen PCOLR0 (704; $2C0) die Hintergrundfarbe enthält.

713,714 $2C9,$2CA RUNADR (X)
715,716 $2CB,$2CC HIUSED (X)
717,718 $2CD,$2CE ZHIUSE (X)
Zwischenregister für den Parameterblock, nur von der Gerätetreiberladeroutine verwendet.

719,720 $2CF,$2D0 GBYTEA (X)
721,722 $2D1,$2D2 LOADAD (X)
723,724 $2D3,$2D4 ZLOADA (X)
Zwischenregister für die Gerätetreiberladeroutine.

725,726 $2D5,$2D6 DSCTLN (X)
Diese beiden Bytes geben die Länge eines Diskettensektors an. Sie werden von der DSKINV-Routine (58451; $E453) beim STATUS-Befehl gesetzt. Bei den ATARI-Diskettenlaufwerken 810 und 1050 ist die Länge eines Sektors 128 ($80) Bytes. Bei Laufwerken mit doppelter Schreibdichte kann sie 256 ($100) Bytes betragen.

727,728 $2D7,$2D8 ACMISR (X)
Diese beiden Speicherstellen wurden für Testzwecke bei der Systementwicklung verwendet.

729 $2D9 KRPDEL (X)
Ansprechzeit für die Tastenwiederholfunktion; bei PAL-Geräten wird dieses Register auf 40 ($28) initialisiert.

730 $2DA KEYREP (X)
Geschwindigkeit der Tastenwiederholfunktion; bei PAL-Geräten auf 5 initialisiert.

731 $2DB NOCLIK (X)
Flag für den Tastaturklick. Hierbei bedeutet der Wert 0, dass nach jedem Tastendruck ein Klickgeräusch erzeugt wird, 255 ($FF) hingegen, dass dies nicht geschieht. Die An- und Abschaltung des Klickgeräusches kann bei dem mit Funktionstasten ausgestatteten ATARI 1200XL und bei XL/XE-Computern mit nachgerüsteten Funktionstasten über die Tastenkombination CONTROL-F3 vorgenommen werden. Da diese Funktion einen eigenen Code in der Tastenfunktionstabelle besitzt (137; $89), lässt sie sich durch Eintragen dieses Wertes in eine im RAM liegende Tastenfunktionstabelle (→ Keyboard) auch auf eine andere Taste legen.

732 $2DC HELPFG (X)
Dieses Statusregister gibt an, ob die HELP-Taste gedrückt worden ist, und wenn ja, in Verbindung mit welchen anderen Tasten dies geschehen ist.

Nachfolgend die entsprechende Wertetabelle:

Dez	Hex	Funktion
0	$0	HELP-Taste nicht gedrückt.
17	$11	nur HELP-Taste gedrückt.
81	$51	HELP und SHIFT gedrückt.
145	$91	HELP und CONTROL gedrückt.

Die HELP-Taste liefert gemeinsam mit SHIFT und CONTROL gedrückt keinen Wert. Nach einer erfolgten Abfrage sollte dieses Register durch Einschreiben des Wertes 0 gelöscht werden.

733 $2DD DMASAV (X)
Zwischenregister für SDMCTL (559; $22F). Dieses Register wird beim ATARI 1200XL und mit Funktionstasten nachgerüsteten XE/XE-Computern benötigt, um den momentanen Wert von SDMCTL zu retten, wenn über die Tastenkombination CONTROL-F2 der DMA abgeschaltet wird.

734 $2DE PBPNT (X)
Zeiger auf das nächste zu übertragende Zeichen im Druckerpuffer. 0 entspricht also der Anfangsadresse des Puffers. Der Wert liegt zwischen 0 und PBUFSZ.

735 $2DF PBUFSZ (X)
Länge des Druckerpuffers. Wenn PBPNT und PBUFSZ den gleichen Wert enthalten, wird der Inhalt des Druckerpuffers ausgedruckt. PBUFSZ wird auf 40 ($28) initialisiert.
Die folgenden vier Bytes werden vom DOS verwendet.

736,737 $2E0,$2E1 RUNAD
Startadresse, zu der nach dem Laden eines Maschinenprogramms gesprungen wird. Unter DOS kann diese Adresse als Datenblock bei der Option SAVE BINARY FILE dem Programm angefügt werden.

738,739 $2E2,$2E3 INITAD
Zu dieser Adresse wird beim Laden eines Maschinenprogramms gesprungen, sobald diese beiden Bytes gesetzt sind. Auch diese Adresse kann beim Speichern als Datenblock hinter das Programm gekettet werden.

740 $2E4 RAMSIZ
Dieses Byte gibt ähnlich wie RAMTOP (106; $6A) die Obergrenze des RAM-Speichers an. Im Gegensatz zu RAMTOP sollte es aber nie verändert werden, um weiterhin die RAM-Obergrenze auf einfache Weise ermitteln zu können.

741,742 $2E5,$2E6 MEMTOP
Zeiger auf das letzte für den Anwender freie Byte im Speicher. Oberhalb davon liegen nur noch die Display-List und die Bildspeicherdaten. Der Wert von MEMTOP ändert sich beim Wechsel der Grafikstufe. Siehe auch APPMHI (14,15; $E,$F).

743,744 $2E7,$2E8 MEMLO

Als Gegensatz zu MEMTOP findet sich hier die Adresse des ersten für den Anwender freien Bytes im Speicher. Ist kein DOS oder RS232-Treiber für das 850-Interfacemodul geladen, ist sie 1792 ($700). Durch das DOS oder durch die Treiber wird MEMLO entsprechend höher gelegt. Siehe auch LOMEM (128,129; $80,$81).

745 $2E9 HNDLOD (X)

Flag für die Gerätetreiberladeroutine.

746-749 $2EA-$2ED DVSTAT

Diese vier Bytes dienen dazu, den Status externer Geräte wiederzugeben und werden infolge eines STATUS-Kommandos in der DSKINV-Routine (58451; $E453) gesetzt. Die Bedeutung der Register im Einzelnen:

Die Speicherstelle 746 ($2EA) enthält den allgemeinen Fehlerstatus des externen Gerätes sowie einige Informationen speziell für Diskettenlaufwerke. Die einzelnen Bits haben in diesem Fall, sofern sie gesetzt sind, nachstehende Funktion:

Bit	Dez	Hex	Funktion
7	128	$80	Die eingelegte Diskette ist in Medium Density formatiert.
6	64	$40	Die XF551 wurde auf Quad Density umgestellt.
5	32	$20	Die eingelegte Diskette ist in Double Density formatiert.
4	16	$10	Der Motor der Diskettenstation läuft noch (nicht bei XF551).
3	8	$8	Die eingelegte Disk ist schreibgeschützt (bei XF551 nur nach Schreibversuch gültig).
2	4	$4	Das letzte Schreib-/Lesekommando war nicht erfolgreich.
1	2	$2	Der letzte Datenblock war fehlerhaft.
0	1	$1	Der letzte Kommandoblock war fehlerhaft.

Die Speicherstelle 747 ($2EB) enthält den Status des Geräts, dessen genaue Bedeutung vom jeweiligen Gerät abhängt. Bei Diskettenlaufwerken steht hier das Status-Byte des Floppy-Disk-Controllers (FDC).

Das Register 748 ($2EC) enthält den maximalen Timeout-Wert des Gerätes; er entspricht etwa Sekunden. Siehe auch DSKTIM (582; $246).

Die Speicherstelle 749 ($2ED) ist nicht definiert. Meist steht hier 0.

750,751 $2EE,$2EF CBAUDL/H

Dieses 2-Byte-Register gibt die Übertragungsrate (Baudrate) des Kassettenrecorders an. Es wird auf 600 Bit/Sekunde (600 Baud) initialisiert und durch die SIO je nach Motorschwankungen usw. korrigiert.

752 $2F0 CRSINH

Ist der Wert in dieser Speicherstelle 0, so ist der Cursor sichtbar. Bei jedem anderen Wert ist er ausgeschaltet. Eine Änderung dieses Registers wird erst nach einer Cursorbewegung wirksam. Durch Drücken von BREAK oder RESET, Änderung der Grafikstufe oder Öffnung eines I/O-Kanals auf den Bildschirm (S:) oder Editor (E:) wird das Register gelöscht, d.h., der Cursor wird wieder sichtbar.

753 $2F1 KEYDEL

Dieses Register dient zur Tastaturentprellung. Ist eine Taste gedrückt, so wird die Zahl 3 in das Register geschrieben. Durch den SYSTEM-Vertikal-Blank-Interrupt (2. Teil) wird dieser Zähler jede 1/50 Sekunde dekrementiert, bis er 0 erreicht. Wird eine Taste neu gedrückt, während der Zähler noch größer als 0 ist, so wird dieser Tastendruck als Prellen interpretiert und ignoriert.

754 $2F2 CH1

Diese Speicherstelle enthält den vorhergehenden Tastaturcode: übernommen aus CH (764; $2FC). Eine Tabelle dieser Codes findet sich im Tabellenteil. Falls zweimal hintereinander dieselbe Taste gedrückt wird, d.h. CH und CH1 gleich sind, so muss KEYDEL (753; $2F1) 0 sein, damit der Tastendruck angenommen wird. Dies dient zur Entprellung der Tastatur.

755 $2F3 CHACT

Durch dieses Register wird die Anzeigeweise aller Zeichen festgelegt. Die folgende Übersicht macht dies deutlich:

Bit	Funktion
0	Inverse Zeichen werden als Leerzeichen dargestellt.
1	Inverse Zeichen werden invers dargestellt.
2	Alle Zeichen werden auf dem Kopf dargestellt.

Mit dem Ausdruck "Inverse Zeichen" sind die Zeichen im Bildspeicher gemeint, deren höchstwertiges Bit gesetzt ist, die also invers dargestellt werden. Durch Kombination der Bits erhält man unterschiedliche Darstellungen. Dieses Register wird auf den Standardwert 2 ($2) initialisiert. CHACT ist das Schattenregister zu CHACTL (54273; $D401).

756 $2F4 CHBAS
Dieses sehr wichtige Register ist die Basisadresse des Zeichensatzes, die Adresse also, von der an aufwärts das Aussehen der Zeichen auf dem Bildschirm definiert wird. Zeichensätze müssen in der Grafikstufe 0 auf 1KByte-Grenzen anfangen, in GRAPHICS 1 und 2 auf geraden Seitengrenzen beginnen; die Anfangsadresse eines Zeichensatzes muss also durch 512 ($200) teilbar sein. In CHBAS wird dann nur die Seitennummer (das High-Byte der Adresse) abgespeichert. CHBAS, welches das Schattenregister zu CHBASE (54281; $D409) ist, wird durch Neuinitialisierung bzw. Änderung der Grafikstufe oder durch das Drücken von RESET auf 224 ($E0), das der Adresse des ROM-Zeichensatzes entspricht, zurückgesetzt. Der internationale Zeichensatz kann bei den XL/XE-Geräten durch Einschreiben des Wertes 204 ($CC) eingeschaltet werden. Die einzelnen Zeichen sind im Übrigen nicht in der ATASCII-Reihenfolge, sondern in der "internen" Reihenfolge im Zeichensatz abgespeichert. Eine Tabelle dieser internen Reihenfolge der Zeichen findet sich im Tabellenteil.

757 $2F5 NEWROW (X)
Zeile, zu der ein DRAWTO- oder FILL-Befehl hinzielt.

758,759 $2F6,$2F7 NEWCOL (X)
Spalte, zu der ein DRAWTO- oder FILL-Befehl hinzielt.

760 $2F8 ROWINC (X)
761 $2F9 COLINC (X)
Diese beiden Speicherstellen enthalten das Vorzeichen (+1 oder -1) der Register DELTAR (118; $76) bzw. DELTAC (119,120; $77,$78).

762 $2FA CHAR
Dieses Register gibt den "internen" Code des zuletzt ein- oder ausgegebenen Zeichens an. Da dies aber meist der Cursor ist, steht hier fast immer der Wert 128 ($80).

763 $2FB ATACHR
Der ATASCII-Code des zuletzt über die CIO ein- oder ausgegebenen Zeichens bzw. Punktes findet sich hier. Außerdem bestimmt dieses Register die Farbe, mit der das DRAW-Kommando arbeitet.

764 $2FC CH

In diesem Register steht der Tastaturcode der zuletzt gedrückten Taste. Der Tastaturtreiber wird ausschließlich über dieses Register betrieben. Der Code der einzelnen Tasten setzt sich wie folgt zusammen: Bit 7 ist gesetzt, wenn zusätzlich CONTROL gedrückt ist, Bit 6 wird gesetzt, wenn dies bei SHIFT der Fall ist. Die restlichen fünf Bits enthalten den Tastaturcode, für den eine Tabelle im Tabellenteil zu finden ist. Falls dieses Register zu einer Tastaturabfrage eingesetzt werden soll, ist es günstig, nach der Abfrage die Speicherstelle durch Einschreiben des Wertes 255 ($FF) zu löschen. Siehe auch CH1 (754; $2F2).

765 $2FD FILDAT

Hier wird die Farbe für das FILL-Kommando abgelegt.

766 $2FE DSPFLG

Ist diese Speicherstelle 0, so werden alle Steuerzeichen normal interpretiert, ist sie jedoch ungleich 0, so wird das Drücken und Anzeigen der Steuerzeichen so behandelt, als ob jedes Mal vorher ESCAPE gedrückt worden wäre. Es werden also bei Steuerzeichen nur die entsprechenden Zeichen ausgegeben, jedoch keine Steuerfunktion ausgeführt. Das gilt nicht für EOL (155, $9B).

767 $2FF SSFLAG

Dieses Register wird durch das Drücken von CONTROL-1 abwechselnd auf 255 ($FF) und 0 gesetzt. Beim Wert 255 ($FF) wird die Bildschirmausgabe gestoppt, beim Wert 0 ist alles normal.

1.4 Seite 3 bis 5 (Page 3 – 5)

Die dritte Seite des Speichers bis Speicherstelle 959 ($3C0) enthält die notwendigen Informationen, um I/O-Operationen durchführen zu können.

Dez	Hex	Funktion
768-779	$300-$30B	Gerätekontrollblock ("Device Control Block", DCB). Er wird dazu verwendet, um mit externen Geräten über die SIO Daten austauschen zu können.
780-793	$30C-$319	Diese Register haben verschiedene Verwendungen.
794-831	$31A-$33F	Adressen der Treiber (Handler) für die einzelnen Geräte. Jeder Handler (E:, S:, C:, P:, D:, R:) besteht aus mehreren Routinen, mit denen ein spezielles Gerät angesprochen werden kann. Eigene Handler können hinzugefügt werden.
832-959	$340-$3BF	Dies sind die acht Ein-/Ausgabekontrollblöcke (IOCB), die für die CIO und die einzelnen Handler wichtig sind, da sie die notwendigen Daten enthalten, um mit den externen Geräten kommunizieren zu können.

Genaue Informationen über die SIO und CIO finden sich in den entsprechenden Abschnitten im Nachschlagteil.

768 $300 DDEVIC
Dieses Byte dient zur Identifikation des Gerätetyps. Die einzelnen Gerätetypen haben folgende Codes:

Dez	Hex	Gerät
49	$31	Diskettenstation
64	$40	Drucker
80	$50	RS232-Schnittstelle (nur im Interfacemodul ATARI 850)
96	$60	Kassettenrecorder

769 $301 DUNIT
Dieses Register enthält die Gerätenummer des externen Gerätes, dabei sind Werte zwischen 1 und 8 möglich. Gerätetyp und Gerätenummer identifizieren das anzusprechende externe Gerät.

770 $302 DCOMND
Das Kommando, das ausgeführt werden soll. Die nachfolgende Tabelle verdeutlicht die Funktion der einzelnen Kommandos:

Dez	Hex	ASCII	Funktion
33	$21	!	Formatieren einer Diskette.
34	$22	"	Formatieren einer Diskette im Format "Medium Density" der 1050.
78	$4E	N	("iNto"). Bei Stationen mit doppelter Schreibdichte wird der sogenannte "Configuration Block" in den Computer gelesen.
79	$4F	O	("Out"). Bei Diskettenstationen mit doppelter Schreibdichte wird der Configuration Block in die Diskettenstation geschrieben.
80	$50	P	("Put"). Schreiben auf die Diskette ohne Prüfung der geschriebenen Daten. Dieses Kommando wird von der DSKINV-Routine (58451; $E453) im 400/800er nicht unterstützt. Der Einsprung DSKINV (58451; $E453) kann deshalb nicht verwendet werden!
82	$52	R	("Read"). Lesen von Daten.
83	$53	S	("Status"). Das externe Gerät (außer Kassettenrecorder) meldet dann den Betriebszustand. Siehe auch DVSTAT (746-749; $2EA-$2ED).
87	$57	W	("Write"). Schreiben von Daten - bei Diskettenlaufwerken mit Verify.

771 $303 DSTATS
Nach einer SIO-Operation wird hier ein eventueller Fehlercode abgelegt. Ist ein Fehler aufgetreten, so ist Bit 7 gesetzt. Eine Tabelle der Fehlercodes findet sich im Tabellenteil. Außerdem wird dieses Register dazu verwendet, um festzulegen, ob es sich um eine Datenausgabe- (Bit 7 gesetzt) oder um eine Dateieingabeoperation (Bit 6 gesetzt) handelt. Dies muss aber vor dem SIO-Einsprung (SIOV: 58457; $E459) festgelegt werden. Das gleichzeitige Setzen von Bit 7 und 6 ist zwar möglich, wird aber von keinem Gerät unterstützt.

772,773 $304,$305 DBUFLO/HI
Dieser Vektor zeigt auf die Adresse im Speicher, an die oder von der Daten übertragen werden. Bei Kommandos, die keine Datenübertragung beinhalten, muss dieses Register nicht gesetzt werden.

774 $306 DTIMLO
Timeout-Wert für das externe Gerät. Er entspricht in etwa Sekunden.

775 $307 DUNUSE
Dieses Register ist zwar augenblicklich unbenutzt, kann aber in späteren Betriebssystem-Versionen belegt werden.

776,777 $308,$309 DBYTLO/HI
Dies Register gibt die Zahl der Bytes an, die übertragen werden soll.

778,779 $30A,$30B DAUX1/2
Zusätzliche Informationen für die speziellen Geräte können in diesen beiden Hilfsbytes abgelegt werden. Im Falle der Benutzung der Diskettenstation steht hier zum Beispiel die anzusprechende Sektornummer.

780,781 $30C,$30D TIMER1
Hier wird die Startzeit beim Messen der Baudrate für das Lesen eines Kassettenblocks abgelegt.

782 $30E ADDCOR (A)
Diese Speicherstelle enthält den Korrekturwert für die Baudrate.

782 $30E JMPERS (X)
Eine Verwendung durch das Betriebssystem ist nicht feststellbar, eine eigene Benutzung ist nicht empfehlenswert.

783 $30F CASFLG
Alle SIO-Operationen, die den Kassettenrecorder verwenden, setzen dieses Register auf einen Wert ungleich von 0.

784,785 $310,$311 TIMER2
Hier wird die Endzeit beim Messen der Baudrate für das Lesen eines Kassettenblocks abgelegt.

Die Bytes 786, 787 und 789 ($312, $313 und $315) werden von den Gerätetreiberladeroutinen benutzt.

788 $314 PTIMOT (X)
Die Werte in diesem Register entsprechen in etwa dem Zeitraum in Sekunden, in dem der Drucker eine positive Statusmeldung an den Computer gesandt haben muss. Ansonsten wird I/O-Fehler 138 (Time Out!) gemeldet. Dieses Register wird durch die Druckertreibersoftware bei jedem STATUS-Befehl neu initialisiert.

790 $316 SAVIO

Diese Speicherstelle enthält das letzte empfangene Bit beim Messen der Kassetten-Baudrate, d.h. Bit 4 von SKSTAT (53775; $D20F).

791 $317 TIMFLG

0 bedeutet, dass der vom SIO gesetzte Timeout (d.h. die maximale Antwortzeit) für das angesprochene SIO-Gerät abgelaufen ist.

792 $318 STACKP

Direkt nach dem Einsprung in die SIO wird der Stapelzeiger hier abgespeichert und vor dem Verlassen der SIO von hier zurück übernommen.

793 $319 TSTAT

Dieses Register speichert zeitweise den Wert von STATUS (48; $30).

794-831 $31A-$33F HATABS (A)
794-828 $31A-$33C HATABS (X)

Diese Tabelle enthält die Adressen der Tabellen für die einzelnen Handler (Treiber). Das Format der Handleradressentabelle sieht wie folgt aus: Zuerst steht der ATASCII-Code des Handlers (zum Beispiel: C,D,E,K,P,S,R), dann die Adresse der Tabelle für den Treiber. Drei Bytes sind also für jeden Handler reserviert. Leere Einträge sind durch Nullen gekennzeichnet.

Die Vektortabelle eines Handlers sieht nun wie folgt aus:

Offset	Funktion
0	Adresse der OPEN-Routine -1.
2	Adresse der CLOSE-Routine -1.
4	Adresse der GET-BYTE-Routine -1.
6	Adresse der PUT BYTE-Routine -1.
8	Adresse der GET-STATUS-Routine -1.
10	Adresse der SPECIAL-Routine -1.
12	Sprung zur Initialisierungsroutine (JMP INIT).
15	Dieses Byte ist 0 (unbenutzt).

Mit "Offset" ist die Zahl gemeint, die zum Anfang der Handler-Tabelle hinzuaddiert werden muss, um auf die entsprechende Adresse zu kommen. Die Offsets 12 und 15 werden vom Betriebssystem nicht unterstützt.

Die Handler-Adressen-Tabelle wird durch das Drücken von RESET auf folgende fünf Handler neu initialisiert:

Dez	Hex	Name	Adresse der Handlertabelle
794	$31A	P:	58416; $E430
797	$31D	C:	58432; $E440
800	$320	E:	58368; $E400
803	$323	S:	58384; $E410
806	$326	K:	58400; $E420

DOS legt die Vektoren DOSVEC (10,11; $A,$B) und DOSINI (12,13; $C, $D), die ja nach dem Drücken von RESET automatisch durchsprungen werden, sodass beim Durchspringen dieser beiden Vektoren der Handler für die Diskette (D:) an die Tabelle angefügt wird.

829 $33D PUPBT1 (X)
830 $33E PUPBT2 (X)
831 $33F PUPBT3 (X)
Mithilfe dieser drei Bytes stellt die Initialisierungsroutine des Betriebssystems der XL/XE-Geräte fest, ob es sich um einen Kalt- oder Warmstart handelt (→ Bootroutine).

Die nächsten 128 ($80) Bytes sind für die acht Ein-/Ausgabekontrollblöcke (IOCB0-7) reserviert. Für jeden IOCB sind sechzehn Bytes vorgesehen. Die einzelnen IOCBs werden wie folgt verwendet: IOCB0 (832-847; $340-$34F) ist normalerweise für den Bildschirmeditor (E:) belegt. Er kann daher von BASIC aus nicht geschlossen werden. Wird eine Grafikstufe angewählt, so wird Kanal 6 (IOCB6: 928-943; $3A0-$3AF) für den Grafikbereich (S:) geöffnet, das Textfenster ist weiterhin über Kanal 0 ansprechbar. Kanal 7 wird für die BASIC-Kommandos LIST, ENTER, SAVE, LOAD und LPRINT verwendet. Weitere Informationen über die CIO und die IOCBs finden sich im jeweiligen Abschnitt im Nachschlagteil.

832-847 $340-$34F IOCB0
848-863 $350-$35F IOCB1
864-879 $360-$36F IOCB2
880-895 $370-$37F IOCB3
896-911 $380-$38F IOCB4
912-927 $390-$39F IOCB5
928-943 $3A0-$3AF IOCB6
944-959 $3B0-$3BF IOCB7

Dies sind die Adressbereiche der 8 "Input/Output Control Blocks" (IOCB). Jeder der IOCBs ist nach dem Beispiel von IOCB #0 organisiert:

Byte Nummer	IOCB #0 in Dez	IOCB #0 in Hex	Register
1	832	$340	ICHID
2	833	$341	ICDNO
3	834	$342	ICCOM
4	835	$343	ICSTA
5 & 6	836/837	$344/$345	ICBADR
7 & 8	838/839	$346/$347	ICPUT
9 & 10	840/841	$348/$349	ICBLEN
11	842	$34A	ICAX1
12	843	$34B	ICAX2
13	844	$34C	ICAX3
14	845	$34D	ICAX4
15	846	$34E	ICAX5
16	847	$34F	ICAX6

960-999 $3C0-$3E7 PRNBUF
Der Druckerpuffer des Drucker-Gerätetreibers. Er wird auf dem Drucker ausgegeben, wenn er mit 40 ($28) Zeichen gefüllt ist oder ein EOF (End Of File, Carriage Return, ATASCII-Wert 155 bzw. Hex-Wert $9B) gesendet wird.

1000 $3E8 SUPERF (X)
Flag des Tastatur-Treibers. Ungleich 0 bedeutet, dass die Funktionstasten F1-F4 zusammen mit Shift gedrückt wurden und damit statt normaler Cursorbewegungen ihre Spezialfunktionen (Home, Bottom, Left Margin, Right Margin) auszulösen sind.

1001 $3E9 CKEY (X)
Während des Kaltstarts wird vom Computer die START-Taste abgefragt; falls sie gedrückt ist, wird CKEY auf einen Wert ungleich von 0 gesetzt und versucht, von der Kassette zu booten.

1002 $3EA CASSBT (X)
Falls erfolgreich von der Kassette gebootet wurde, wird in dieses Register ein Wert ungleich von 0 geschrieben. Siehe auch BOOT? (9; $9).

1003 $3EB CARTCK (X)
Prüfsumme des Speicherbereiches von 49136 ($BFF0) bis 49391 ($C0EF). Sie wird benutzt, um festzustellen, ob ein anderes Programmmodul eingelegt worden ist, woraufhin ein Kaltstart ausgelöst wird.

1004 $3EC DEERF (X)
Ungleich 0 bedeutet, dass der Screen-Handler aufgrund von Speichermangel nicht geöffnet werden konnte. Das passiert, wenn APPMHI (14,15; $E,$F) zu hoch und/oder RAMTOP (106; $6A) zu niedrig ist.

1005-1015 $3ED-$3F7 ACMVAR (X)
Für Testzwecke reservierte Bytes, die beim Warmstart nicht gelöscht werden.

1016 $3F8 BASICF (X)
Flag, ob das eingebaute ATARI-BASIC ein- oder ausgeschaltet ist, 0 bedeutet, dass das BASIC eingeschaltet, 1, dass es ausgeschaltet ist.

1017 $3F9 MINTLK (X)
Für interne Testzwecke reserviert.

1018 $3FA GINTLK (X)
Schattenregister zu TRIG3 (53267; $D013). 1 bedeutet, dass ein Programmmodul eingesteckt und aktiv ist, 0 das Gegenteil. Dieses Register wird während des System-Vertical-Blank-Interrupts mit TRIG3 verglichen, um festzustellen, ob ein Modul eingesteckt oder herausgenommen wurde. Bei einer Veränderung wird bei nachfolgendem RESET ein Kaltstart ausgelöst.

1019,1020 $3FB,$3FC CHLINK (X)
Zeiger für die Gerätetreiberladeroutine.

1021-1151 $3FD-$47F CASBUF
Dieser Puffer wird vom Kassettentreiber dazu verwendet, Daten zu empfangen bzw. zu senden. Die eigentlichen Daten, 128 ($80) Bytes, stehen ab der Speicherstelle 1024 ($400). Die drei Bytes am Anfang jedes Kassettenrecords haben die folgende Bedeutung:

1021,1022 $3FD,$3FE ...
Diese beiden Bytes sind immer 85 ($55). Dies ist eine abwechselnde Folge von gesetzten und ungesetzten Bits. Sie dient zur Geschwindigkeitsmessung und hilft bei der Festlegung der Baudrate.

1023 $3FF ...
Dieses Byte kontrolliert die Art des folgenden Records durch diese Werte:

Dez	**Hex**	**Funktion**
250	$FA	Der Record ist weniger als 128 ($80) Bytes lang. Im letzten Byte des Records ist die Länge zu finden.
252	$FC	Normaler Record mit 128 ($80) Bytes.
254	$FE	Letzter Record des Files (EOF). Es folgen 128 ($80) leere Bytes.

Nach jedem Record folgt noch die Prüfsumme des gesamten Records. Dieses Prüfbyte wird aber nicht in den Puffer übernommen.

Außerdem wird in diesen Puffer noch der erste Diskettensektor während des Bootvorgangs gelesen.

Die Speicherstellen 1152-1407 ($480-$57F) sind unbenutzt. Sie sind für die Verwendung durch Anwendungsprogramme (wie z.B. dem BASIC-ROM) vorgesehen. Nutzer dieser Anwendungsprogramme (z.B. ein BASIC-Programm) dürfen diesen Bereich nicht verwenden.

Der Rest der Seite 5 wird von den Fließkommaroutinen verwendet. In Maschinensprache sind auch diese Bytes frei, falls die Fließkommaroutinen nicht verwendet werden.

1406,1407 $57E,$57F LBPR1,2
Diese zwei Bytes werden von der FASC-Routine (55526; $D8E6) zur Ablage des Minuszeichens und der Null vor dem Dezimalpunkt benutzt.

1408-1535 $580-$5FF LBUFF
Wird von verschiedenen Fließkommaroutinen benutzt. Z.B. legt die FASC-Routine (55526; $D8E6) hier ihr Ergebnis ab (in ASCII umgewandelte Fließkommazahl), auf deren Beginn INBUFF (243,244; $F3,$F4) verweist.

1.5 Seite 6 (Page 6)

Die Seite 6 (1536-1791; $600-$6FF) wird vom Betriebssystem und dem normalen ATARI-BASIC nicht benutzt, sie steht dem Benutzer also zur freien Verfügung.

1.6 Seite 7 (Page 7)

Der Bereich 1792-49151 ($700-$BFFF) steht dem Benutzer für seine Programme beliebig zur Verfügung. Das DOS liegt immer direkt von 1792 ($700) an aufwärts und erstreckt sich bis zu der in MEMLO (743,744; $2E7,$2E8) angegebenen Adresse. Besitzt man jedoch nur einen 16K-

Computer (z.B. ATARI 400 oder ATARI 600XL) ist natürlich nur bis zur Speicherstelle 16383 ($3FFF) benutzbares RAM vorhanden.

1.7 Bereich $5000-$57FF

20480 $5000 STORG (X)
Im Bereich von 20480-22527 ($5000-$57FF) können, sofern über PORTB (54017; $D301) eingeschaltet, die Selbsttestroutinen liegen. Der Selbsttest kann über die Adresse PUPDIV (58496; $E480) aufgerufen werden.

Eingelegte Programmmodule legen sich über das RAM, dieser Bereich ist daher ebenfalls nicht mehr verwendbar. Programmmodule können bis zu 16K-Byte lang sein. Die oberste freie Adresse ist dann 32767 ($7FFF).

1.8 Module (Cartridges)

Die Initialisierung der Programmmodule wird durch die letzten sechs Bytes in jedem Modul kontrolliert, die folgende Bedeutung haben:

49146,49147 $BFFA,$BFFB CARTCS
Startadresse des Moduls.

49148 $BFFC CART
Diese Speicherstelle muss 0 sein, wenn das Betriebssystem ein eingestecktes Modul erkennen soll. Ist sie nicht 0, ignoriert das Betriebssystem das Modul, d.h. die Speicherstellen CARTCS, CARTFG, CARTAD werden ignoriert. Ist kein Modul eingelegt, so kann diese Speicherstelle auch Werte ungleich 0 annehmen.

49149 $BFFD CARTFG
Dieses Byte kontrolliert verschiedene Optionen, die, falls die entsprechenden Bits gesetzt sind, zur Initialisierung des Moduls dienen:

Bit	Dez	Hex	Funktion
7	128	$80	Bei Reset wird ohne jede Initialisierung sofort per 6502-JMP über den Vektor CARTAD in das Modul gesprungen.
2	4	$4	Ist das Bit gesetzt, wird das Modul initialisiert (CARTAD) und gestartet (CARTCS). Sonst wird es nur initialisiert (CARTAD).
0	1	$1	Flag, ob die Peripheriegeräte gebootet werden sollen.

49150,49151 $BFFE,$BFFF CARTAD
Initialisierungsadresse des Moduls. Ist Bit 7 von CARTFG gesetzt, wird beim Reset ohne jede Initialisierung durch diesen Vektor gesprungen. Ansonsten wird dieser Vektor per 6502-JSR beim Reset direkt vor Öffnen des Editors aufgerufen.

1.9 Bereich $C000-$CFFF

Der Speicherblock von 49152 bis 53247 ($C000-$CFFF) wird bei den XL/XE-Geräten als Betriebssystem-ROM verwendet. Über PORTB (54017; $D301), das die Speicherverwaltung (MMU) steuert, kann jedoch auch das RAM, das darunter liegt, eingeschaltet werden.

Im Bereich von 52224-53247 ($CC00-$CFFF) liegt bei den XL/XE-Geräten, sofern das Betriebssystem-ROM eingeschaltet ist, der internationale Zeichensatz. Dieser Zeichensatz kann durch Einschreiben des Wertes 204 ($CC) in das Register CHBAS (756; $2F4) eingeschaltet werden. Er entspricht weitgehend dem normalen ATARI-Zeichensatz, enthält jedoch anstelle der Grafikzeichen verschiedene internationale Zeichen wie deutsche Umlaute, skandinavische, spanische und französische Sonderzeichen.

1.10 Hardwareregister

Im Bereich von 53248 bis 55295 ($D000-$D7FF) liegen die sogenannten Hardwareregister. Es sind die Register für die vier zusätzlichen Chips (GTIA, POKEY, PIA, ANTIC) der ATARI-Computer. Diese Register sind nicht wie normales RAM zu benutzen, da man zwar Werte in sie hineinschreiben kann, beim Lesen dieser Register jedoch andere Werte erhält. Vor jeder einzelnen Beschreibung steht deshalb ein '-S-' oder '-L-' (Schreiben/Lesen), das anzeigt, in welcher Richtung das Register zu verwenden ist. Viele der Hardwareregister haben sogenannte Schattenregister, die man stattdessen wie normales RAM verwenden kann. Der zweite Teil des System-Vertikal-Blank-Interrupts überträgt diese Schattenregister in die Hardwareregister, sofern sie '-S-'-Register sind, bzw. liest die Hardwareregister und schreibt sie in die entsprechenden Schattenregister, falls sie '-L-'-Register sind. Falls nun ein Hardwareregister permanent geändert werden soll, so muss nur in das Schattenregister der entsprechende Wert geschrieben werden, die Übertragung in das Hardwareregister wird vom Betriebssystem durchgeführt. Soll jedoch während des Bildschirmaufbaus, z. B. durch einen DLI, eine Farbe oder ein Zeichensatz umgeschaltet werden, muss natürlich direkt in das Hardwareregister geschrieben werden. Die Schattenregister zu den entsprechenden Hardwareregistern stehen in Klammern hinter der '-L-'- bzw. '-S-'-Angabe.

1.10.1 GTIA

Der Speicherbereich von 53248 bis 53505 ($D000-$D0FF) ist zum GTIA-Chip (Graphic Television Interface Adapter) verdrahtet und enthält daher dessen Hardwareregister.

53248 $D000 HPOSP0
53249 $D001 HPOSP1
53250 $D002 HPOSP2
53251 $D003 HPOSP3
-S- (/)
In diese Register können die horizontalen Positionen der linken Kanten der Player 0 bis 3 abgespeichert werden. Dabei sind Werte von 0 bis 255 ($0-$FF) möglich, sichtbar sind jedoch nur Werte von 48 bis 208 ($30-$D0). Diese Zahlenangaben hängen jedoch von der Darstellungsweise des verwendeten Fernsehers/Monitors ab und können daher variieren. Da jeder Player 128 ($80) bzw. 256 ($100) Bytes "hoch" ist, müssen die vertikalen Bewegungen softwaremäßig erzeugt werden. Sichtbar sind hier wiederum auch nur die Objekte innerhalb der Positionen 16 bis 112 ($10-$70) bzw. 32 bis 224 ($20-$E0). Diese Angaben gelten für die doppelzeilige bzw. einzeilige Auflösung der Player/Missile-Grafik.

53248 $D000 M0PF
53249 $D001 M1PF
53250 $D002 M2PF
53251 $D003 M3PF
-L- (/)
Dies sind die Kollisionsregister von Missile 0 bis 3 mit den verschiedenen Anzeigefeld-Farben (Playfield-Farben). Falls das Missile mit den einzelnen Anzeigefeldern überlappt, werden folgende Bits der betreffenden Register gesetzt:

Bit	Dez	Hex	Kollision mit Anzeigefeld-Farbe Nr.
3	8	$8	3
2	4	$4	2
1	2	$1	1
0	1	$0	0

53252 $D004 HPOSM0
53253 $D005 HPOSM1
53254 $D006 HPOSM2

53255 $D007 HPOSM3
-S- (/)
In diese vier Register kann jeweils die horizontale Position der Missiles geschrieben werden. Siehe HPOSP0-3 (53248-53251; $D000-$D003).

53252 $D004 P0PF
53253 $D005 P1PF
53254 $D006 P2PF
53255 $D007 P3PF
-L- (/)
Diese Register geben die Kollisionen der vier Player mit den verschiedenen Anzeigefeld-Farben an. Die Bits werden genauso verwendet und gesetzt wie in den Registern M0-3PF (53248-53251; $D000-$D003).

53256 $D008 SIZEP0
53257 $D009 SIZEP1
53258 $D00A SIZEP2
53259 $D00B SIZEP3
-S- (/)
In diesen Registern kann die Breite der einzelnen Player festgelegt werden. Es sind dabei drei verschiedene Breiten möglich. Die Bits jedes Players werden dabei auf ein, zwei oder vier Farbpunkte ausgedehnt, d.h., es ist normale, doppelte oder vierfache Breite der Objekte möglich.

Die folgenden Werte müssen in dieses Register geschrieben werden, um die entsprechenden Playerbreiten zu erhalten:

Bit	Dez	Hex	Breite
0,1	3	$3	vierfach
0,1	2	$2	normal
0,1	1	$1	doppelt
0,1	0	$0	normal

53256 $D008 M0PL
53257 $D009 M1PL
53258 $D00A M2PL
53259 $D00B M3PL
-L- (/)
Dieses Register stellt die Kollision der Missiles 0 bis 3 mit den einzelnen Playern fest. Bei Überlappung werden folgende Bits gesetzt:

Bit	Dez	Hex	Kollision mit Player Nr.
3	8	$8	3
2	4	$4	2
1	2	$2	1
0	1	$1	0

53260 $D00C SIZEM
-S- (/)
Dieses Register legt die Breiten der einzelnen Missiles fest. Es sind dabei dieselben Breiten wie bei SIZEP0-3 (53256-53259; $D008-$D00B) möglich. Die Bits sind mit den einzelnen Missiles wie folgt verknüpft:

Bit	Missile Nr.
7,6	3
5,4	2
3,2	1
1,0	0

Die einzelnen Bits müssen folgendermaßen gesetzt werden, um die verschiedenen Breiten zu erhalten:

Bitkombination	Breite
11	vierfach
10	normal

Bitkombination	Breite
01	doppelt
00	normal

53260 $D00C P0PL
53261 $D00D P1PL
53262 $D00E P2PL
53263 $D00F P3PL
-L- (/)
Diese vier Register geben die Kollisionen der einzelnen Player untereinander genauso wie in M0-3PL (53256-53259; $D008-$D00B) an.

53261 $D00D GRAFP0
53262 $D00E GRAFP1
53263 $D00F GRAFP2
53264 $D010 GRAFP3

-S- (/)
Durch diese vier Register ist es möglich, ohne Einschaltung des ANTIC direkt durch den GTIA Grafik mit den Playern und Missiles zu erzeugen. Das Byte in diesen Registern wird als senkrechter Balken über den gesamten Bildschirm dargestellt. Die Farben und Positionen werden wie gewohnt eingestellt.

53264 $D010 TRIG0
53265 $D011 TRIG1
53266 $D012 TRIG2 (A)
53267 $D013 TRIG3 (A)
-L- (STRIG0-3: 644-647; $284-$287)
Diese vier Register geben den Status der Controllerknöpfe an. Sie sind 0, wenn der Knopf gedrückt ist, und 1, wenn der Knopf nicht gedrückt ist. Es ist nun zusätzlich möglich, durch Setzen von Bit 2 von GRACTL (53277; $D01D) das Lesen der Knöpfe umzuschalten. In diesem Fall bleibt, wenn ein Knopf gedrückt und wieder losgelassen wird, das zugehörige Hard-wareregister so lange 0, bis Bit 2 von GRACTL zurückgesetzt wird.

53266 $D012 TRIG2 (XEGS)
-L- (/)
Über TRIG2 kann das XEGS feststellen, ob die Tastatur angeschlossen ist. In diesem Fall steht hier 1, sonst 0. Das Betriebssystem nutzt dies beim Kaltstart, um bei nicht angeschlossener Tastatur das eingebaute Missile Command über PB6 der PIA zu aktivieren.

53267 $D013 TRIG3 (X)
-L- (GINTLK: 1018; $3FA)
Über TRIG3 kann festgestellt werden, ob ein Modul eingelegt ist. In diesem Fall nimmt es den Wert 1 an, ansonsten ist es 0.

53265 $D011 GRAFM
-S- (/)
Diese Speicherstelle ermöglicht, ähnlich wie GRAFP0-3 (53261-53264; $D00D-$D010), Player/Missile-Grafik direkt über den GTIA zu erzeugen. Für jede Missile stehen hier wie folgt zwei Bits zur Verfügung:

Bit	Missile Nr.
7,6	3
5,4	2
3,2	1
1,0	0

Farben und horizontale Positionen werden wie üblich gesetzt.

53266 $D012 COLPM0
53267 $D013 COLPM1
53268 $D014 COLPM2
53269 $D015 COLPM3
-S- (PCOLR0-3: 704-707; $2C0-$2C3)
Dies sind die vier Farbregister für Player und Missiles 0 bis 3.

53268 $D014 PAL
-L- (/)
Dieses Register erlaubt es dem Programm, zu unterscheiden, ob es auf einem PAL-ATARI (das Register ist dann 1) oder einem NTSC-ATARI läuft. In diesem Fall sind die Bits 1 bis 3 gesetzt, der Wert des Registers ist dann 14 ($E).

53270 $D016 COLPF0
53271 $D017 COLPF1
53272 $D018 COLPF2
53273 $D019 COLPF3
53274 $D01A COLBK
-S- (COLOR0-4: 708-712; $2C4-$2C8)
Die Farben auf dem Bildschirm in den vier Anzeigefeldern und im Hintergrund werden durch diese fünf Farbregister angegeben. Genauere Informationen zu diesen Registern finden sich bei den entsprechenden Schattenregistern.

53275 $D01B PRIOR
-S- (GPRIOR: 623; $26F)
Dieses Register steuert die Rangfolge der anzuzeigenden Objekte, d.h. ob bei Überlappungen die Player ober- oder unterhalb der Anzeigefeld-Farben dargestellt werden sollen. Es ermöglicht weiterhin das Einschalten der verschiedenen GTIA-Modi (GRAPHICS 9 bis 11), die Darstellung eines fünften Players durch die vier Missiles und die Erzeugung mehrfarbiger Player. Genauere Informationen zu diesem Register finden sich beim entsprechenden Schattenregister.

53276 $D01C VDELAY
-S- (/)
Ist die zweizeilige Auflösung der Player und Missiles eingeschaltet, muss die Möglichkeit bestehen, Objekte nur über einzelne Bildschirmzeilen zu bewegen. Eine solche Möglichkeit ist durch dieses Register gegeben. Wird das entsprechende Bit gesetzt, so wird das betreffende Objekt um genau eine Bildschirmzeile nach unten bewegt.

Bit	Dez	Hex	Objekt
7	128	$80	Player 3
6	64	$40	Player 2
5	32	$20	Player 1
4	16	$10	Player 0

Bit	Dez	Hex	Objekt
3	8	$08	Missile 3
2	4	$04	Missile 2
1	2	$02	Missile 1
0	1	$01	Missile 0

Eine Verschiebung um mehr als eine Bildschirmzeile muss durch Speicherbewegungen erfolgen.

53277 $D01D GRACTL
-S- (/)
Dieses Register kontrolliert die Player und Missiles und erlaubt es, Joystick- oder Paddleknopfeingaben zu "speichern". Zu dieser "Speicherung" finden sich nähere Informationen bei TRIG0-3 (53264-53276; $D010-D013). Die Tabelle zeigt die Verwendungsweise der einzelnen Bits:

Bit	Dez	Hex	Funktion
2	4	$4	Das Drücken der Knöpfe wird gespeichert.
1	2	$2	Player an/aus
0	1	$1	Missiles an/aus

53278 $D01E HITCLR
-S- (/)
Durch das Einschreiben eines beliebigen Wertes werden sämtliche Kollisionsregister gelöscht. Dies bietet sich in Spielen an, nachdem die Kollisionsregister abgefragt sind. Bis zur nächsten Abfrage (die z. B. im Vertikal-Blank-Interrupt geschehen kann) können dann erneut Kollisionen registriert werden. Es ist auf jeden Fall wichtig, die Kollisionsregister regelmäßig zu löschen.

53279 $D01F CONSOL
-S/L- (/)
Der Status der drei Zusatztasten (OPTION, SELECT, START) kann in diesem Register abgefragt werden. Die entsprechenden Bits werden gelöscht, wenn die betreffende Taste gedrückt wird. Der Normalinhalt dieses Registers ist 7. Bits: 2 – OPTION, 1 – SELECT, 0 – START.

Beim Schreiben bewirken die Bits Folgendes:

Bit	Funktionen
3	Lautsprecher-Position: 0=aktiv, 1=inaktiv
2	1=Taste OPTION gesperrt → OPTION-Abfrage liefert immer 0
1	1=Taste SELECT gesperrt → SELECT-Abfrage liefert immer 0
0	1=Taste START gesperrt → START-Abfrage liefert immer 0

Im 2. Teil des System-Vertikal-Blank-Interrupts wird hierhin 8 geschrieben, um die Konsolentaster abfragen zu können, womit auch der Lautsprecher zurück in die Normalposition gesetzt wird. Bit 3 wird auch für die Erzeugung des Tastaturklicks benutzt.

Die Speicherstellen 53280 bis 53503 ($D020 bis $D0FF) wiederholen die vorstehenden Speicherstellen.

Im Bereich zwischen 53504 und 53759 ($D100-$D1FF) liegen die Register zur Bedienung des parallelen Busses. Obwohl es keine offizielle technische Dokumentation zum Parallelbus gibt, konnten die relevanten Informationen zusammengetragen werden. Ihr findet die Details im Kapitel 2 (→ Reset und Bootvorgang bzw. → Parallelbusgeräte).

53504 $D100 PDVIN (X)
53504 $D100 PDVOUT (X)
-S/L- (/)
Datenein- und -ausgaberegister für den parallelen Bus.

53505 $D101 PDVRST (X)
-S- (/)
Durch Einschreiben eines beliebigen Wertes wird PDVIN initialisiert.

53505 $D101 PDVSTA (X)
-L- (/)
Statusregister für den parallelen Bus.

53759 $D1FF PDVS (X)
-S- (SHPDVS: 584; $248)
Geräteauswahlregister für den parallelen Bus. Siehe auch SHPDVS.

53759 $D1FF PDVIRQ (X)
-L- (/)
Interrupt-Statusregister für den parallelen Bus.

1.10.2 POKEY

Der POKEY-Chip (POtentiometer and KEYboard Controller Chip) ist für die gesamte Tonerzeugung, die Abfrage der Paddles, das Lesen der Tastatur und für die Dateiein- und -ausgabe über den seriellen Port verantwortlich. Zur eingehenden Information finden sich zu den einzelnen oben genannten Themen Abschnitte im Nachschlagteil. Um den POKEY-Chip zu initialisieren, muss 0 in AUDCTL (53768; $D208) und 3 in SKCTL (53775; $D20F) geschrieben werden. Bei den XL/XE-Ataris wird beim Reset zwar 3 in SKCTL geschrieben, aber 32 in AUDCTL. Auch nach SIO-Operationen ist AUDCTL nicht 0. Daher muss man vor der Nutzung der Sound-Register AUDCTL auf 0 setzen, sonst klingt der Sound meist nicht richtig.

53760 $D200 AUDF1
53762 $D202 AUDF2
53764 $D204 AUDF3
53766 $D206 AUDF4
-S- (/)
Diese vier Register steuern die Frequenz der vier Tonkanäle. In diese Speicherstellen wird eine Zahl X geschrieben. Von der anliegenden Eingangsfrequenz wird nur jeder X-te Impuls ausgegeben. Je größer also der Wert dieser Speicherstellen wird, desto niedriger ist die hörbare Frequenz. Der Wert in diesen Registern ist der zweite Parameter des SOUND-Kommandos in BASIC. Außerdem können diese Speicherstellen als Zähler verwendet werden. Informationen dazu finden sich bei VTIMR1 (528,529; $210,$211).

53761 $D201 AUDC1
53763 $D203 AUDC2
53765 $D205 AUDC3
53767 $D207 AUDC4
-S- (/)
Diese Register kontrollieren die Tonerzeugung der vier Tonkanäle. Die Bits in diesen Registern haben folgende Bedeutung:

Bit	Funktion
07-05	Verzerrung (siehe zweite Tabelle)
4	Volume-Only an/aus
3-0	Lautstärke (Werte zwischen 0 und 15 ($F)) sind möglich.

Die Tonerzeugung verläuft wie folgt: Die Eingangsfrequenz (siehe AUDCTL (53768; $D208) wird durch den Wert im AUDF-Register dividiert, anschließend wird das Ergebnis mit verschiedenen Zufallszählern (Polyzählern) maskiert. Zum Schluss wird noch einmal durch zwei geteilt. Die Technik der Tonerzeugung wird in Kapitel 2 → Sound beschrieben.

Die Verzerrungsbits haben folgende Wirkung:

Bit			**Funktion**
7	**6**	**5**	
0	0	0	5-Bit-, 17-Bit-Polyzähler
0	X	1	Nur 5-Bit-Polyzähler
0	1	0	5-Bit-, 4-Bit-Polyzähler
1	0	0	Nur 17-Bit-Polyzähler
1	X	1	reiner Rechteckton (keine Polyzähler)
1	1	0	Nur 4-Bit-Polyzähler

X bedeutet, dass der Wert dieses Bits nicht relevant ist.

53760 $D200 POT0
53761 $D201 POT1
53762 $D202 POT2
53763 $D203 POT3
53764 $D204 POT4
53765 $D205 POT5
53766 $D206 POT6
53767 $D207 POT7
-L- (PADDL0-7: 624-631; $270-$277)
Hier finden sich die acht (XL/XE-Geräte: nur die ersten vier sind relevant) Register, die den Wert der Paddles angeben. Das direkte Lesen dieser Hardwareregister ist insofern schwierig, als die Inhalte der Register nicht in jedem Fall richtig sind. Der korrekte Lesevorgang sieht wie folgt aus:

- Starten der Auslesesequenz, die während des Ablaufs von 228 ($E4) Bildschirmzeilen stattfindet, durch Einschreiben eines beliebigen Wertes in POTGO (53771; $D20B).

- Warteschleife, bis ALLPOT (53768; $D208) den Wert 0 hat.

- In diesem Moment sind die Werte gültig.

Für alle normalen Anwendungen reicht eigentlich jedoch die Benutzung der Schattenregister, die immer stimmen.

53768 $D208 AUDCTL
Dieses Register bietet zusätzliche Steuermöglichkeiten der Klangerzeugung. Folgende Tabelle macht dies deutlich:

Bit	Dez	Hex	Funktion
7	128	$80	Der 17-Bit-Polyzähler wird zum 9-Bit-Polyzähler.
6	64	$40	Kanal 1 hat nun als Basisfrequenz 1,77 MHz (normal: 64 KHz).
5	32	$20	Kanal 3 hat nun als Basisfrequenz 1,77 MHz.
4	16	$10	16-Bit-Auflösung für Kanal 1 und 2.
3	8	$08	16-Bit-Auflösung für Kanal 3 und 4.
2	4	$04	Höhenfilter für Kanal 1, durch Kanal 2 gesteuert.
1	2	$02	Höhenfilter für Kanal 3, durch Kanal 4 gesteuert.
0	1	$01	Als Basisfrequenz wird nun 15 KHz verwendet.

53768 $D208 ALLPOT
Status für den Wert der POT0-7 Register (53760-53767; $D200-$D207). Für jedes Register ist das zugehörige Bit 0, wenn der Wert gültig ist. Das Register muss also insgesamt 0 sein, wenn die gesamten Paddlewerte in Ordnung sein sollen. Bei den XL/XE-Geräten sind nur die unteren vier Bits wichtig, die oberen vier Bits sollten deshalb beim Lesen dieses Registers ausmaskiert werden. Siehe auch POTGO (53771; $D20B).

53769 $D209 STIMER
-S- (/)
Um die vorhandenen POKEY-Zähler (bei 400/800 zwei, bei XL/XE-Geräten drei) zu starten, wird irgendeine Zahl in dieses Register geschrieben. Informationen zu den POKEY-Zählern finden sich bei VTIMR1 (528,529; $210,$211).

53769 $D209 KBCODE
-L- (CH: 764; $2FC)
Liefert den Tastaturcode der momentan gedrückten Taste. Er wird während des Tastaturinterrupts in CH übertragen. Bit 7 ist gesetzt, wenn CONTROL gedrückt ist, Bit 6, wenn SHIFT gedrückt ist (→ Keyboard).

53770 $D20A SKREST
-S- (/)
Das Einschreiben eines beliebigen Wertes in dieses Register führt dazu, dass die oberen drei Bits von SKCTL (53775; $D20F) (5, 6 und 7) auf 1 gesetzt werden.

53770 $D20A RANDOM
-L- (/)
Dieses Register enthält die oberen acht Bits des 17-Bit-Polyzählers, somit kann diese Speicherstelle als Zufallszahlengenerator (Werte zwischen 0 und 255 ($FF)) verwendet werden.

53771 $D20B POTGO
-S- (/)
Wird ein beliebiger Wert in dieses Register geschrieben, so werden die Register POT0-7 (53760-53767; $D200-$D207) auf 0 gesetzt. Anschließend (während des Aufbaus von 228 Bildschirmzeilen) werden diese Register neu gelesen. Sie haben dann den endgültigen Wert, wenn das entsprechende Bit in ALLPOT (53768; $D208) auf 0 gesetzt ist.

53772 $D20C ...
Dieses POKEY-Register ist unbenutzt und kann nicht verändert werden.

53773 $D20D SEROUT
-S- (/)
Dieses Register enthält das Byte, das gerade über den seriellen Port übertragen werden soll. Es wird durch den POKEY-Chip bitweise während des "Serial output data needed"-Interrupts übertragen.

53773 $D20D SERIN
-L- (/)
Bits, die vom seriellen Port übernommen werden, werden so lange in dieses Register geladen, bis ein Byte komplett ist. Dieses Register wird dann während des "Serial input data ready"-Interrupts verarbeitet.

53774 $D20E IRQEN
-S- (POKMSK: 16; $10)
Die maskierbaren Interrupts (IRQ) werden über dieses Register ein- und ausgeschaltet. Jedes Bit dieses Registers aktiviert, sofern es gesetzt ist, einen bestimmten Interrupt. Weitere Informationen finden sich beim Schattenregister POKMSK. Um den POKEY-Chip zu initialisieren (er hat keine RESET-Leitung), muss der Wert 3 in dieses Register und der Wert 0 in AUDCTL (53768; $D208) geschrieben werden.

53774 $D20E IRQST
-L- (/)
Dieses Register ermöglicht es festzustellen, welche Ursache einen IRQ ausgelöst hat. Die Bits werden daher ebenso wie bei IRQEN verwendet, nur ist dasjenige Bit nicht gesetzt, das den Interrupt ausgelöst hat.

53775 $D20F SKCTL
-S- (SSKCTL: 562; $232)
Über dieses Register kann der serielle Port kontrolliert werden. Weitere Informationen finden sich beim Schattenregister SSKCTL.

53775 $D20F SKSTAT
-L- (/)
Diese Speicherstelle enthält den Status der seriellen Schnittstelle. Die einzelnen Bits dieses Registers sind normalerweise gesetzt. Wenn sie 0 sind, enthalten sie die nachstehenden Informationen:

Bit	Dez	Hex	Funktion
7	128	$80	Fehlende oder überschüssige Bits in der seriellen Daten-übertragung. ("serial data input frame error").
6	64	$40	Überlagerung bei der Dateneingabe über die Tastatur, dadurch fehlt mindestens ein Tastendruck. ("keyboard overrun").
5	32	$20	Überlagerung bei der Dateneingabe über den seriellen Port, dadurch fehlt mindestens ein Bit. ("serial data input overrun").
4	16	$10	Momentaner Zustand der Dateneingangsleitung des SIO-Anschlusses.
3	8	$08	SHIFT-Taste ist gedrückt.
2	4	$04	Die zuletzt gedrückte Taste ist weiterhin gedrückt.
1	2	$02	Das serielle Schieberegister arbeitet noch.
0	1	$01	Dieses Bit ist immer gesetzt.

Die Bits 5 bis 7 bleiben, sofern sie einmal gelöscht worden sind, so lange 0, bis sie durch Einschreiben eines beliebigen Wertes in SKREST (53770; $D20A) wieder gesetzt werden.

Die Speicherstellen 53776 bis 54015 ($D210-$D2FF) wiederholen aufgrund der unvollständigen Adressdecodierung die Speicherstellen 53760 bis 53775 ($D200-$D20F). Dieser Bereich wird durch den zweiten POKEY bei Stereo-Erweiterungen genutzt.

1.10.3 PIA

Der PIA (Peripheral Interface Adapter), ein 6520-Chip (gelegentlich auch 6521, 6820, 6821), steuert die Controllerports und bei den XL/XE-Geräten den Speicherverwaltungschip ("Memory Management Unit", MMU).

Die Register dieses Standardchips sind im A8 in der Reihenfolge abweichend von der sonst üblichen Verwendung wie folgt verdrahtet:

54016 $D300 PORTA
-S/L- STICK0 & 1 (632,633; $278,$279) und
PTRIG0-3 (636-639; $27C-$27F)
Mit Hilfe dieses Registers ist es möglich, Daten über die Controllerports 1 und 2 aus- und einzugeben, zusätzlich wird hier festgelegt, ob Daten über die Ports gelesen oder geschrieben werden. Die Bits 0 bis 3 sind mit den Pins 1 bis 4 des ersten Ports verknüpft, die Bits 4 bis 7 mit den Pins 1 bis 4 des zweiten Ports. Normalerweise ist dieses Register auf Eingabe geschaltet. Es liest dann z.B. die Joystickwerte. Wird PORTA als Eingaberegister verwendet, so haben die einzelnen Bits dieses Register folgende Funktionen:

Bit	Port	Joystick	Paddleknopf von Paddle Nr.
7	2	Rechts	3
6	2	Links	2
5	2	Rückwärts	/
4	2	Vorwärts	/
3	1	Rechts	1
2	1	Links	0
1	1	Rückwärts	/
0	1	Vorwärts	/

Die Joysticknummer entspricht dabei der Portnummer. Die Bits sind normalerweise gesetzt, werden jedoch beim Drücken der Paddleknöpfe bzw. beim Bewegen der Joysticks gelöscht.

Sollen jedoch Daten ausgegeben werden, muss zuvor Bit 2 von PACTL durch Einschreiben von 48 ($30) gelöscht und in PORTA 255 ($FF) eingeschrieben werden. Anschließend muss Bit 2 von PACTL wieder durch Einschreiben von 52 ($34) gesetzt werden. Nun können Daten über die

Ports ausgegeben werden. Um wieder auf Eingabe zu schalten, muss bei gelöschtem Bit 2 von PACTL "0" in PORTA geschrieben werden.

54017 $D301 PORTB (A)
-S/L- STICK2 & 3 (634,635; $27A,$27B) und
PTRIG4-7 (640-643; $280-$283)

Beim ATARI 400/800 hat PORTB für die Controllerports 3 und 4 die gleiche Funktion wie PORTA für Controllerport 1 und 2. Port B wird dann von PBCTL gesteuert.

54017 $D301 PORTB (X)
-S/L- (/)

Beim XL/XE wird dieses Register für die Verwaltung von Erweiterungsspeicher eingesetzt. Allerdings gibt es dafür keinen einheitlichen Standard. Einige Varianten bis 256 KByte nutzen den 1985 von ATARI eingeführten '130XE-Standard' und werden allgemein als Typ "CS" (= Compy Shop) bezeichnet. Sie erlauben den getrennten Zugriff von CPU und ANTIC auf den Erweiterungsspeicher. Andere Modelle und alle Erweiterungen über 256 KByte bis hin zu 4 MByte werden üblicherweise als Typ "Rambo" (= Rambo XL) bezeichnet. Bei ihnen greifen CPU und ANTIC nur gemeinsam auf den gleichen Speicher zu. Welches Modell welche Bits im Port B des PIA wie nutzt, muss aus der Hardwaredokumentation des jeweiligen Herstellers entnommen werden.

Daraus folgt, dass nicht jede Speichererweiterung mit jedem DOS in vollem Umfang als RAM-Disk verwendet werden kann. Was darüber hinaus möglich ist, hängt von der verwendeten Software ab.

Belegung dieses Registers beim ATARI 1200XL (64 KByte):

Bit 0	gesetzt:	$C000-$FFFF → Betriebssystem-ROM
	nicht gesetzt:	$C000-$FFFF → RAM
Bit 1	unbenutzt	
Bit 2	gesetzt:	LED 1 ein
	nicht gesetzt:	LED 1 aus
Bit 3	gesetzt:	LED 2 ein
	nicht gesetzt:	LED 2 aus
Bit 4-6	unbenutzt	
Bit 7	gesetzt:	$5000-$57FF → RAM
	nicht gesetzt:	$5000-$57FF → Selbsttest-ROM

Belegung beim ATARI 600XL/800XL (64 KByte):

Bit 0 gesetzt: $C000-$FFFF → Betriebssystem-ROM
nicht gesetzt: $C000-$FFFF → RAM

Bit 1 gesetzt: $A000-$BFFF → RAM
nicht gesetzt: $A000-$BFFF → ATARI-BASIC-ROM

Bit 2-6 unbenutzt

Bit 7 gesetzt: $5000-$57FF → RAM
nicht gesetzt: $5000-$57FF → Selbsttest-ROM

Belegung beim ATARI 130XE (128 KByte):

Bit 0 gesetzt: $C000-$FFFF → Betriebssystem-ROM
nicht gesetzt: $C000-$FFFF → RAM

Bit 1 gesetzt: $A000-$BFFF → RAM
nicht gesetzt: $A000-$BFFF → ATARI-BASIC-ROM

Bit 2-3: Nummer der zusätzlichen RAM-Bank
im Bereich von $4000-$7FFF

Bit 2,3	Bank
0 0	0
0 1	1
1 0	2
1 1	3

Bit 4 gesetzt: CPU hat Zugriff auf das normale RAM.
nicht gesetzt: CPU hat Zugriff auf das zusätzliche RAM.

Bit 5 gesetzt: ANTIC hat Zugriff auf das normale RAM.
nicht gesetzt: ANTIC hat Zugriff auf das zusätzliche RAM.

Bit 6 unbenutzt

Bit 7 gesetzt: $5000-$57FF → RAM
nicht gesetzt: $5000-$57FF → Selbsttest-ROM

Weitere Hinweise zur Verwendung des zusätzlichen Speichers der XL/XE-Geräte finden sich im Kapitel 2 → Speicheraufteilung.

54018 $D302 PACTL
-S/L- (/)
Dieses Register kontrolliert den Port A des PIA und somit die Controller-ports 1 und 2. Die Bits in diesem Register haben folgende Funktionen:

Bit	Dez	Hex	Funktion
7	128	$80	Dieses Bit kann nur gelesen werden! Es gibt den Status des Interrupts der "Proceed Line" der seriellen Schnittstelle an (1=Interrupt, durch Lesen von PORTA wird dieses Bit zurückgesetzt).
6	64	$40	Dieses Bit ist immer 0.
5	32	$20	Dieses Bit ist immer 1.
4	16	$10	Dieses Bit ist immer 1.
3	8	$8	Motor des Kassettenrecorders ein- (0) bzw. ausschalten (1).
2	4	$4	1=PORTA wird zur Datenein- oder -ausgabe verwendet, 0=Es wird die Schreib-/Leserichtung durch Einschreiben eines Wertes in PORTA festgelegt. Das "direction control register" des PIA wird angesprochen. (1=Schreiben, 0=Lesen über den betreffenden Pin des Ports)
1	2	$2	Dieses Bit ist immer 0.
0	1	$1	Der Interrupt der "Proceed Line" ist ein-/ausgeschaltet. Dieses Bit wird vom Betriebssystem auf 0 gesetzt.

Das Betriebssystem setzt dieses Register auf 60 ($3C). Bei angeschaltetem Kassettenmotor steht hier 52 ($34).

54019 $D303 PBCTL (A)
-S/L- (/)
Beim ATARI 400/800 hat dieses Byte dieselbe Funktion wie PACTL, bis auf zwei Bits: Bit 3 kontrolliert den Zustand der Kommando-Leitung des SIO-Anschlusses. Bit 7 gibt den Interrupt-Status der sogenannten Interrupt-Leitung am SIO-Anschluss wieder. Vom Betriebssystem wird dieses Byte auf 60 ($3C) gesetzt.

Die Speicherstellen 54020 bis 54271 ($D304-$D3FF) wiederholen aufgrund unvollständiger Adressdekodierung die vier vorstehenden Speicherstellen.

1.10.4 ANTIC

Der ANTIC-Chip (AlphaNumeric Television Interface Controller) ist wohl der "intelligenteste" Koprozessor der 6502-Zentraleinheit der ATARI-Computer. Er kontrolliert den GTIA-Chip und liefert diesem durch direkten Speicherzugriff ("Direct Memory Access", DMA) die benötigten Daten. Er ist selbst in einfacher Weise programmierbar. Eine Tabelle der Programmbefehle des ANTIC findet sich im Tabellenteil.

54272 $D400 DMACTL
-S- (SDMCTL: 559,$22F)
Dieses Register kontrolliert den direkten Speicherzugriff, schaltet Player und Missiles ein und aus und kontrolliert die Breite des Anzeigefelds. Die Funktionen der einzelnen Bits sind bei SDMCTL dokumentiert.

54273 $D401 CHACTL
-S- (CHACT: 755; $2F3)
Dieses Register steuert die Darstellungsweise der Zeichen in allen Textgrafikstufen. Wie die einzelnen Bits verwendet werden, steht bei CHACT.

54274,54275 $D402,$D403 DLISTL/H
-S- (SDLSTL: 560,561; $230,$231)
Dieses Register gibt die Anfangsadresse der Display List an. Sie ist das "Programm" für den ANTIC. Weitere Hinweise zum ANTIC und zur Display List finden sich in eigenen Abschnitten im Nachschlagteil und bei SDLSTL.

54276 $D404 HSCROL
-S- (/)
Dieses Register gibt die Zahl der Farbpunkte ("color clocks") an, um die diejenigen Bildzeilen nach rechts verschoben werden, bei denen im entsprechenden ANTIC-Befehl Bit 4 gesetzt ist. Hierbei sind Werte zwischen 0 und 15 möglich. Damit lässt sich sehr einfach das sogenannte Fein-Scrolling erreichen, ohne dass irgendwelche zeitaufwendigen Speicherverschiebungen notwendig sind. Details zum Fein-Scrolling finden sich im Nachschlagteil → ANTIC.

54277 $D405 VSCROL
-S- (/)
In dieses Register kann die Zahl der Farbpunkte geschrieben werden, um die diejenigen Bildzeilen nach oben verschoben werden, deren entsprechende ANTIC-Befehle Bit 5 gesetzt haben. Hier sind Werte zwischen 0 und 7, bei sechzehn Bildschirmzeilen hohen Grafikstufen (GRAPHICS 2 und 13) Werte zwischen 0 und 15 möglich. Details zum Fein-Scrolling befinden sich im Kapitel 2 → ANTIC.

Bei XL/XE-Geräten kann der komplette Textbildschirm auf Fein-Scrolling geschaltet werden, indem FINE (622; $26E) auf einen Wert ungleich von 0 gesetzt und anschließend der Bildschirm neu geöffnet wird.

54278 $D406 ...
Dieses Register ist nicht belegt.

54279 $D407 PMBASE
-S- (/)
In dieses Register wird die Seitenzahl des Beginns des Speicherbereichs für die Player/Missile-Grafik geschrieben. Hierbei sind nur 1KByte-Grenzen (zweizeilige Auflösung) bzw. 2KByte-Grenzen (einzeilige Auflösung) möglich. Wie die Speicheraufteilung für die einzelnen Player/Missiles aussieht, wird aus der Tabelle im Tabellenteil deutlich. Zusätzlich existiert ein Abschnitt über die Player/Missile-Grafik im Tabellenteil.

54280 $D408 ...
Dieses Register ist nicht belegt.

54281 $D409 CHBASE
-S- (CHBAS: 756; $2F4)

Dieses Register enthält die Basisadresse des Zeichensatzes. Hier wird nur das High-Byte (die Seitennummer) abgespeichert. In GRAPHICS 0, 12 und 13 muss der Zeichensatz auf einer 1KByte-Grenze beginnen, er ist 128*8=1024 Bytes lang. In den Grafikstufen 1 und 2 ist der Zeichensatz nur 64*8=512 Bytes lang, er muss daher auf einer geraden Seitenzahl beginnen. Siehe auch CHBAS und der Abschnitt über Zeichensätze im Nachschlagteil.

54282 $D40A WSYNC
-S- (/)
Wird eine beliebige Zahl in diese Speicherstelle geschrieben, so wird die 6502-CPU solange gestoppt, d.h., es geschieht nichts, bis der Elektronenstrahl der Bildröhre das Ende der Bildschirmzeile erreicht. Dies dient dazu, z.B. bei der Anwendung von DLIs zur Umschaltung von Farben ein Flackern zu vermeiden. Dazu muss nur vor der Änderung des Farbregisters ein beliebiger Wert in WSYNC geschrieben werden. Bei den 400/800er Geräten verwendet die "Klick"-Routine der Tastatur ebenfalls WSYNC zur Verzögerung.

54283 $D40B VCOUNT
-L- (/)
Dieses Register enthält die Nummer der momentan erzeugten Bildschirmzeile geteilt durch zwei. Die Werte liegen bei europäischen PAL-

Geräten zwischen 0 und 155 ($9B), bei amerikanischen NTSC-Computern zwischen 0 und 130 ($82).

54284 $D40C PENH
-L- (LPENH: 564; $234)
Dieses Register enthält die horizontale Position des Lightpens.

54285 $D40D PENV
-L- (LPENV: 565; $235)
Dieses Register enthält den Wert von VCOUNT, wenn der Knopf des Lightpens gedrückt wird, gibt somit die vertikale Position des Lightpens an.

54286 $D40E NMIEN
-S- (/)
Auch die nicht maskierbaren Interrupts können bei den ATARI-Rechnern ein- und ausgeschaltet werden. Es sind dies der "Vertikal Blank Interrupt", der "Display List Interrupt" und der "RESET Interrupt", der durch Drücken der RESET-Taste ausgelöst wird. Wenn folgende Bits gesetzt sind, sind die entsprechenden Interrupts möglich:

Bit	Dez	Hex	Interrupt
7	128	$80	Display List Interrupt
6	64	$40	Vertikal Blank Interrupt
5	32	$20	RESET-Interrupt (nicht maskierbar)
4-0			nicht belegt

Den RESET-Interrupt gibt es nur bei den 400/800er Geräten. Er kann nie gesperrt werden. Durch das Betriebssystem wird nur der VBI ermöglicht; will man auch DLIs erzeugen, so muss man den Wert 192 ($C0) in dieses Register schreiben. Siehe auch die Abschnitte über Interrupts, den ANTIC und VBIs im Nachschlagteil.

54287 $D40F NMIRES
-S- (/)
Das Einschreiben eines beliebigen Wertes setzt NMIST zurück. Bit 7 wird vom ANTIC beim Vertical Blank automatisch zurückgesetzt.

54287 $D40F NMIST
-L- (/)
Dieses Register wird verwendet, um die Ursache eines nicht maskierbaren Interrupts festzustellen. Die Bits in dieser Speicherstelle werden dabei ebenso wie in NMIEN (54286; $D40E) verwendet.

Die Speicherstellen 54288 bis 54257 ($D410-$D4FF) wiederholen die sechzehn vorhergehenden Speicherstellen.

Der Bereich von 54258 bis 54783 ($D500-D5FF) ist zum Modulschacht verdrahtet. Module, die die "Cartridge Control"-Leitung verwenden, können z.B. ein- und ausgeschaltet werden oder auch zwischen Speicherbänken im Modul umschalten.

Der Speicherbereich von 54784 bis 55039 ($D600-D6FF) ist für künftige Erweiterungen reserviert.

Der Speicherbereich von 55040 bis 55295 ($D700-$D7FF) ist für interne Testzwecke reserviert.

1.10.5 Fließkommaroutinen

Der Bereich von 55296 bis 57343 ($D800-$DFFF) enthält die Fließkommaroutinen. Details dazu im Kapitel 2 → Fließkomma-Arithmetik.

55296 $D800 AFP
Mithilfe dieser Routine werden ASCII-Zahlen in Fließkommaregister 0 (FR0: 212-217; $D4-$D9) übertragen. Auf den Eingabepuffer wird durch INBUFF (243,244; $F3,$F4) + CIX (242; $F2) gezeigt. Kann eine Umwandlung nicht erfolgen, so wird nach Rückkehr das Carry-Flag gesetzt.

55526 $D8E6 FASC
Umwandlung der Fließkommazahl in FR0 in ASCII-Format. Der Ausgabepuffer liegt dort, wohin INBUFF (243,244; $F3,$F4) zeigt. Er liegt innerhalb des Bereichs LBPR1/2,LBUFF (1406-1535; $57E-$5FF). Bit 7 des letzten Zeichens ist gesetzt.

55722 $D9AA IFP
Die 2-Byte-Integerzahl (Wert von 0 bis 65535) in den Speicherstellen 212 und 213 ($D4,$D5) wird als Fließkommazahl in FR0 abgelegt.

55762 $D9D2 FPI
Umwandlung von FR0 in eine Integerzahl (Speicherstellen 212,213; $D4,$D5). Kann eine Umwandlung nicht erfolgen, da die Zahl in FR0 negativ oder größer als 65535 ist, wird das Carry-Flag gesetzt.

55876 $DA44 ZFR0
Löschung von FR0.

55878 $DA46 ZFR1
Löschung von FR1 (224-229; $E0-$E5).

55904 $DA60 FSUB
Subtraktion: FR0-FR1. Das Ergebnis wird in FR0 gespeichert. Gerät das Ergebnis außerhalb des darstellbaren Bereiches, so wird bei Rückkehr aus der Routine das Carry-Flag gesetzt.

55910 $DA66 FADD
Addition: FR0 + FR1. Das Ergebnis wird in FR0 gespeichert. Gerät das Ergebnis außerhalb des darstellbaren Bereiches, so wird bei Rückkehr aus der Routine das Carry-Flag gesetzt.

56027 $DADB FMUL
Multiplikation: FR0 * FR1. Das Ergebnis wird in FR0 gespeichert. Gerät das Ergebnis außerhalb des darstellbaren Bereiches, so wird bei Rückkehr aus der Routine das Carry-Flag gesetzt.

56104 $DB28 FDIV
Division: FR0/FR1. Das Ergebnis wird in FR0 gespeichert. Gerät das Ergebnis außerhalb des darstellbaren Bereiches, so wird bei Rückkehr aus der Routine das Carry-Flag gesetzt.

56640 $DD40 PLYEVL
Diese Routine rechnet Polynome aus. Beim Einsprung muss das X- und Y-Register (Y-Register als High-Byte) auf die Liste der Koeffizienten A(i), die zuerst die "höherwertigen" Koeffizienten enthalten muss, zeigen. Der Akkumulator gibt die Zahl n der Koeffizienten an. Das Register FR0 enthält schließlich die Variable X, die in das Polynom eingesetzt werden soll. Das Ergebnis (P(x)) der Polynomberechnung wird in FR0 abgespeichert. Folgende Formel wird dabei angewandt:

$$P(x)=SUM(i=0 \text{ bis } n-1) (A(i)*x^i)$$

Kann das Polynom nicht berechnet werden, so ist bei der Rückkehr aus der Routine das Carry-Flag gesetzt.

56713 $DD89 FLD0R
FR0 wird mit der durch das X- und Y-Register bezeichneten Fließkommazahl geladen. Das Y-Register enthält das High-Byte der Adresse.

56717 $DD8D FLD0P
FR0 wird mit der durch das Register FLPTR (252,253; $FC,$FD) bezeichneten Fließkommazahl geladen.

56728 $DD98 FLD1R
FR1 wird mit der durch das X- und Y-Register bezeichneten Fließkommazahl geladen. Das Y-Register enthält das High-Byte der Adresse.

56732 $DD9C FLD1P
FR1 wird mit der durch das Register FLPTR (252,253; $FC,$FD) bezeichneten Fließkommazahl geladen.

56743 $DDA7 FST0R
FR0 wird in die durch das X- und Y-Register bezeichnete Fließkommazahl gespeichert. Das Y-Register enthält das High-Byte der Adresse.

56747 $DDAB FST0P
FR0 wird in die durch das Register FLPTR (252,253; $FC,$FD) bezeichnete Fließkommazahl gespeichert.

56758 $DDB6 FMOVE
FR0 wird nach FR1 übertragen.

56768 $DDC0 EXP
e^{FR0} wird berechnet (e=2,71828...). Das Ergebnis wird in FR0 abgespeichert. Tritt ein Fehler auf (z. B. Überlauf), so ist nach dem Rücksprung das Carry-Flag gesetzt.

56780 $DDCC EXP10
10^{FR0} wird berechnet. Das Ergebnis wird in FR0 abgespeichert. Tritt ein Fehler auf (z. B. Überlauf), so ist nach dem Rücksprung das Carry-Flag gesetzt.

57037 $DECD LOG
Der natürliche Logarithmus von FR0 zur Basis e (e=2,71828...) wird berechnet. Das Ergebnis wird in FR0 abgespeichert. Tritt ein Fehler auf (z. B. Überlauf), so ist nach dem Rücksprung das Carry-Flag gesetzt.

57041 $DED1 LOG10
Der Logarithmus von FR0 zur Basis 10 wird berechnet. Das Ergebnis wird in FR0 abgespeichert. Tritt ein Fehler auf (z. B. Überlauf), so ist nach dem Rücksprung das Carry-Flag gesetzt.

1.10.6 Fließkommabereich oder Parallelbus

Der Speicherbereich zwischen 55296 und 55551 ($D800-$D8FF), der im Normalfall von den Fließkommaroutinen belegt ist, wird auch vom Treiber für den parallelen Bus benutzt. Dazu werden in dem Moment, in dem ein Gerät am parallelen Bus angesprochen wird, die Fließkommaroutinen abgeschaltet und ein 2 KByte langer Speicherbereich von der am parallelen Bus angeschlossenen Hardware eingeblendet. Die ersten 26 ($1A) Bytes müssen folgende Datentabelle enthalten:

55296,55297 $D800,$D801 PDCKSM (X)
Prüfsumme für den eingeblendeten Speicherbereich.

55298 $D802 PDREVN (X)
Kennnummer der Softwareversion.

55299 $D803 PDID1 (X)
Identifikationsnummer 1: 128; $80.

55300 $D804 PDTYPE (X)
Gerätetyp; dieser muss nicht angegeben werden, da die Art der Codierung bislang nicht bekannt ist.

55301 $D805 PDIOV (X)
Einsprungpunkt für die Ein- und Ausgaberoutinen. Es sollte hier ein JMP-Befehl stehen.

55304 $D808 PDIRQV (X)
Einsprungpunkt für die Interruptroutinen (IRQ). Es sollte hier ein JMP-Befehl stehen.

55307 $D80B PDID2 (X)
Identifikationsnummer 2: 145; $91.

55308 $D80C PDNAME (X)
Hier ist ein ein Byte langer Name des Gerätetreibers im ASCII-Format vorgesehen.

55309 $D80D PDVV (X)
In diesem 16 ($10) Bytes langen Bereich muss die eigentliche Treibertabelle liegen, die wie alle anderen Gerätetreibertabellen, z. B. die des Bildschirmeditors (E:), aufgebaut sein muss. Der Aufbau der Treibertabellen ist im → Kapitel 2 im Abschnitt über Gerätetreiber erklärt.

1.10.7 Standardzeichensatz

Im Bereich von 57344 bis 58367 ($E000-E3FF) ist der Standardzeichensatz zu finden. Das Register CHBAS (756; $2F4) enthält nach dem Einschalten deshalb immer 224 ($E0). Für die Definition jedes Zeichens sind acht Bytes notwendig. Die Reihenfolge der Zeichen im Zeichensatz entspricht nicht der ATASCII-Reihenfolge, sie wird als "interne" Reihenfolge bezeichnet. Siehe Vergleichstabelle → Kapitel 3. Weitere Informationen über Zeichensätze stehen im → Kapitel 2.

1.10.8 Interne Treibertabelle

Im Bereich von 58368 bis 58447 ($E400-$E44F) sind die fünf internen Treibertabellen zu finden. Für jeden Treiber (E:, S:, K:, P:, C:) sind in dieser Reihenfolge 16 ($10) Bytes vorhanden. Da die CIO in diese Routinen springt, indem sie die zwei Bytes auf den Stapel schiebt und anschließend einen RTS-Befehl ausführt, sind diese Adressen in den Vektortabellen um eins niedriger als die tatsächliche Einsprungadresse. Durch den Benutzer kann jedoch dieselbe Sequenz

```
LDA VECTOR+1 ;VECTOR ist ein beliebiger
PHA
LDA VECTOR ;Vektor aus der Sprungtabelle.
PHA
RTS ; Sprung nach VECTOR+1
```

verwendet werden.

Die sechzehn Bytes jedes Treibers sind wie folgt aufgeteilt:

Offset	Adresse
+0	OPEN-Routine -1
+2	CLOSE-Routine -1
+4	GET BYTE-Routine -1
+6	PUT BYTE-Routine -1
+8	GET STATUS-Routine -1
+10	SPECIAL-Routine -1
+12	Sprung zur Initialisierungsroutine (JMP LSB/MSB)
+15	nicht verwendet

Folgende Treibertabellen liegen im Betriebssystem-ROM:

58368 $E400 EDITRV
Treibertabelle für den Bildschirmeditor (E:)

58384 $E410 SCRENV
Treibertabelle für den Bildschirmtreiber (S:)

58400 $E420 KEYBDV
Treibertabelle für den Tastaturtreiber (K:)

58416 $E430 PRINTV
Treibertabelle für den Druckertreiber (P:)

58432 $E440 CASETV
Treibertabelle für den Kassettentreiber (C:)

1.10.9 Einsprungadressen ins Betriebssystem

Gute Programmierer vermeiden um jeden Preis, direkt in Betriebssystem-ROM-Routinen zu springen, da sonst die Kompatibilität der verschiedenen Betriebssystem-Versionen nicht gewährleistet ist. Für alle wichtigen Anwendungen existiert die nachfolgende Sprungtabelle, für die ATARI auch in zukünftigen Betriebssystem-Versionen garantiert. Diese Sprungtabelle befindet sich im Bereich von 58488 bis 58510 ($E450-$E48E). Sie besteht aus folgender Aneinanderreihung von JMP-Befehlen:

58488 $E450 DISKIV
Diese Routine initialisiert die DSKINV-Routine (58451; $E453), indem sie die Sektorlänge DSCTLN (725,726; $2D5,$2D6) auf 128 setzt und den Timeout-Wert für das Formatieren DSKTIM (582; $246) auf $A0 setzt. Da das Betriebssystem diese Routine bei jedem Reset aufruft, muss man sie nicht selbst aufrufen.

58451 $E453 DSKINV
Dieser JMP-Befehl verzweigt zu der Disketten-SIO-Routine. Man kann statt dieses Vektors auch den SIOV-Vektor verwenden, jedoch setzt die DSKINV-Routine einige Parameter wie die Sektorlänge, die Schreib/ Leserichtung (vgl. DSTATS (771; $303)) und den Timeout-Wert, der ja für den Formatierungsbefehl erhöht werden muss. Achtung! Das Betriebssystem der 400/800er Geräte unterstützt beim Durchsprung durch diese Routine nicht den Schreibbefehl ohne Überprüfung, also das Diskettenkommando Put ("P", 80; $50). Um dieses Kommando anzuwenden, muss man alle Parameter selbst setzten und einen Sprung durch den SIO-Vektor veranlassen. Weitere Details dazu im Kapitel 2 → SIO.

58454 $E456 CIOV
Dieser JMP-Befehl springt zur zentralen Ein/Ausgaberoutine ("Central Input/Outputroutine", CIO). Das X-Register enthält die Nummer des anzusprechenden Ein/Ausgabeblocks (IOCB) multipliziert mit 16 ($10). Die IOCBs liegen im Speicherbereich zwischen 832 und 959 ($340-$3BF). Vor dem Sprung zu diesem Vektor müssen verschiedene Parameter in den einzelnen IOCBs gesetzt werden. Siehe dazu Kapitel 2 → CIO.

58457 $E459 SIOV

Dieser JMP-Befehl springt zur seriellen Ein/Ausgaberoutine ("Serial Input Output", SIO). Die SIO gibt die Möglichkeit, direkt mit den verschiedenen externen Geräten zu kommunizieren, ohne den Umweg über die CIO, die der SIO praktisch übergeordnet ist, in Kauf nehmen zu müssen. Zur Benutzung der SIO muss der sogenannte Gerätekontrollblock ("Device Control Block", DCB) initialisiert werden. Der DCB liegt im Bereich zwischen 768 und 779 ($300-$30B). Auch für die SIO ist ein eigener Abschnitt im Nachschlagteil vorhanden.

58460 $E45C SETVBV

Der Einsprung in diese Routine ermöglicht es, problemlos verschiedene Interruptvektoren auf der Seite 2 auf den gewünschten Wert zu setzen, ohne dass zum Beispiel in dem Moment, in dem man das Low-Byte des Vektors gesetzt hat, das High-Byte sich jedoch noch auf dem alten Wert befindet, dieser Interrupt ausgelöst wird und ein Absturz des Systems die Folge ist. Man sollte allerdings vor Aufruf der Routine die maskierbaren Interrupts (IRQs) per 6502-SEI sperren, da sonst ein auftretender IRQ doch zum Absturz führen kann.

Vor dem Einsprung in die Routine muss das X-Register der 6502-CPU das High-Byte des neuen Wertes haben, das Y-Register das Low-Byte und der Akku einen der folgenden Werte, um einen der verschiedenen Vektoren auszuwählen:

Akku	Vektor	veränderter Vektor
1	CDTMV1	536,537; $218,$219
2	CDTMV2	538,539; $21A,$21B
3	CDTMV3	540,541; $21C,$21D
4	CDTMV4	542,543; $21E,$21F
5	CDTMV5	544,545; $220,$221
6	VVBLKI	546,547; $222,$223
7	VVBLKD	548,549; $224,$225

58463 $E45F SYSVBV

Dieser JMP-Befehl springt zur System-Vertikal-Blank-Routine. Jeder vom Anwender programmierte "Immediate" VBI, der VBI also, der vor dem System-VBI ausgeführt wird, muss mit einem JMP SYSVBV enden.

58466 $E462 XITVBV

Dieser JMP-Befehl springt zu der Routine, die den VBI verlässt. Sie kehrt in das Hauptprogramm zurück. Jeder vom Anwender programmierte "Deferred" VBI, der VBI also, der nach dem System-VBI ausgeführt wird, muss mit einem JMP XITVBV enden. Der Deferred VBI wird, falls CRITIC (66; $42) einen Wert ungleich 0 hat, also im Moment eine zeitkritische I/O-Operation stattfindet, nicht ausgeführt.

58469 $E465 SIOINV

Diese Routine initialisiert den POKEY-Chip. Das Betriebssystem ruft SIOINV bei jedem Reset auf.

58472 $E468 SENDEV

Diese Routine initialisiert den POKEY-Chip so, dass Daten ausgegeben werden können.

58475 $E46B INTINV

Diese Routine initialisiert das NMIEN-Register (54286; $D40E) auf 64 ($40), was den Vertical Blank-Interrupt aktiviert. Bei XL/XE-Geräten überträgt sie außerdem TRIG3 (53267; $D013) in GINTLK (1018; $3FA). Bei 400/800er-Geräten initialisiert sie zusätzlich alle Ports auf Eingabe.

58478 $E46E CIOINV

Diese Routine dient zur Initialisierung der CIO. Sie setzt den Status aller IOCBs auf "Frei".

58481 $E471 BLKBDV

An diese Stelle wird durch den BASIC-Befehl BYE gesprungen. Bei den 400/800er Geräten ist es ein Sprung in den sogenannten MEMOPAD-Modus, bei den XL/XE-Geräten wird in den Selbsttest gesprungen.

58484 $E474 WARMSV

Dieser JMP-Befehl springt in die Warmstartroutine. Ein Sprung hierher entspricht dem Drücken der RESET-Taste bei gelöschtem COLDST-Byte (580; $244).

58487 $E477 COLDSV

An die durch diesen JMP-Befehl angegebene Adresse wird durch den 6502-Prozessor beim Ausführen des Kaltstartes gesprungen. Ein Sprung hierhin entspricht also einem Aus- und Einschalten des Computers. Bei den 400/800er-Geräten ist dieser Wert auch in PVECT (65532,65533; $FFFC,$FFFD) zu finden. Ebenfalls wird hierhin verzweigt, wenn COLDST (580; $244) einen Wert ungleich 0 hat und RESET gedrückt wird.

58490 $E47A RBLOKV
Diese Routine wird ausschließlich vom Betriebssystem verwendet und liest einen Block (Record) von der Kassette.

58493 $E47D CSOPIV
Diese Routine wird ausschließlich vom Betriebssystem verwendet und öffnet einen Kanal für den Kassettenrecorder zum Einlesen von Daten. Dieser und der vorstehende Vektor werden während der Bootroutine für den Kassettenrecorder verwendet.

Bei den XL/XE-Geräten existieren noch diese fünf weiteren Vektoren:

58496 $E480 PUPDIV (X)
Über diesen Vektor kann der Selbsttest eingeschaltet und gestartet werden. Dieser Einsprung ist dazu gedacht, das sogenannte "Power-Up-Display" aufzurufen und könnte in künftigen Betriebssystemversionen auch eine andere Bedeutung haben.

58499 $E483 SLFTSV (X)
Über diesen Einsprung kann der Selbsttest aufgerufen werden; dazu muss er allerdings zuvor über PORTB eingeschaltet worden sein.

58502 $E486 PHENTV (X)
Diese Routine erleichtert es, einen neuen Eintrag in HATABS (794-828; $31A-$33C) vorzunehmen. Dazu müssen vorher folgende Register gesetzt werden:

X-Register: Gerätename in ATASCII
Akkumulator: Treibertabellenadresse (High-Byte)
Y-Register: Treibertabellenadresse (Low-Byte)

Die Routine überprüft zunächst, ob der Eintrag schon existiert. Ist dies nicht der Fall, wird ein freier Platz gesucht und, sofern möglich, die Eintragung vorgenommen. Nach Rückkehr sind das Carry- und das Negativ-Flag folgendermaßen gesetzt:

Carry: 0: Eintrag vorgenommen
1: Eintrag bereits vorhanden

Negativ: 1: Eintrag nicht vorhanden, jedoch kein Platz für weitere Eintragungen.

58505 $E489 PHULNV (X)
Routine zum Löschen einer Treibertabelle aus der neben HATABS bestehenden Liste der verketteten Treibertabellen.

58508 $E48C PHINIV (X)
Routine zum Laden von relozierbaren CIO-Gerätetreibern über SIO. Die PHINIV-Routine wird beim Kaltstart nach dem Booten von Diskette aufgerufen.

58511 $E48F GPDVV (X)
Allgemeine Treiber-Vektortabelle für sämtliche Peripheriegeräte am PBI.

Auf die Angabe weiterer Einsprünge in das Betriebssystem wird verzichtet, da sich fast alle Adressen der XL/XE-Geräte gegenüber den 400/800er Geräten geändert haben. Außerdem kann ein guter Programmierer auch ohne diese Betriebssystem-Einsprünge auskommen.

65530,65531 $FFFA,$FFFB ...
Diese zwei Bytes enthalten die Adresse der Routine, die die nicht maskierbaren Interrupts (NMIs) verarbeitet.

65532,65533 $FFFC,$FFFD PVECT
Diese zwei Bytes geben die Kaltstartadresse des Computers an. Der 6502-Prozessor springt automatisch an diese Stelle, wenn er ein RESET-Signal erhält. Dieses RESET-Signal wird durch Drücken der RESET-Taste ausgelöst.

65534,65535 $FFFE,$FFFF ...
Diese Adresse zeigt auf die Routine, zu der beim Auftreten eines maskierbaren Interrupts (IRQ) gesprungen wird. In der Routine wird die Ursache des IRQ bestimmt und weitere Unterroutinen ausgeführt.

2 Nachschlagteil

Kapitel 2 wurde ergänzt um wichtige Informationen zu einzelnen Geräten, Speichererweiterungen, Cartridges, DOS, Parallelbus und Bootvorgang.

2.1 ANTIC

Die Abkürzung ANTIC steht für "Alpha Numeric Television Interface Controller", was auf Deutsch etwa alphanumerischer Fernsehkontrollchip bedeutet. Er hat folgende Aufgaben:

- Ausführung und Kontrolle der DMA-Funktionen (DMA: "Direct Memory Access", dt.: direkter Speicherzugriff)

- Kontrolle der nicht maskierbaren Interrupts (NMI: "Non Maskable Interrupt")

- Vertikales und horizontales "Fine scrolling" (feines, ruckelfreies Verschieben von Bildschirmteilen)

- Kontrolle des Lichtgriffels ("Lightpen")

- Zähler für die augenblicklich erzeugte Bildschirmzeile

- Ermöglichung der horizontalen Synchronisation

- Kontrolle des GTIA-Chips

Nun noch zwei Begriffsdefinitionen zum weiteren Verständnis:

- "Color Clock": Mit diesem Begriff ist die kleinstmögliche horizontale Bildeinheit eines Fernsehgerätes gemeint. Eine Bildschirmzeile eines PAL-Gerätes umfasst 228 Color Clocks, von denen jedoch nicht alle vom Anwender angesprochen werden können. Für jeden dieser Color Clocks kann die Farbe beliebig eingestellt werden. Da die größtmögliche Auflösung des ANTIC (ANTIC-Modus 15) über die Auflösung des Fernsehers hinaus geht, kann in diesem Fall die Farbe der Bildelemente nicht mehr eingestellt werden. Effekte wie "artifacting" (Grün- und Blaufärbung von Bildteilen) sind dann möglich.

- "Pixel" ("Picture Cell"): Ein Pixel (deutsch etwa: Bildelement) ist in der jeweiligen Grafikstufe die kleinstmögliche vom Anwender ansprechbare logische Einheit. Die Größe eines Pixels hängt von der gewählten Grafikstufe ab. Sie kann horizontal zwischen 0,5 und 4

Color Clocks und vertikal zwischen einer und acht Bildschirmzeilen liegen.

Um die Funktionsweise des ANTIC zu verstehen, muss zunächst einmal der Aufbau eines analogen Fernsehbildes erklärt werden:

Analoge Fernsehbilder werden zeilenweise durch einen Elektronenstrahl sichtbar gemacht. Dieser wird am hinteren Ende der Bildröhre erzeugt und kann durch Magneteinwirkung auf jeden beliebigen Punkt der Bildschirmoberfläche gelenkt werden. Der Strahl bringt in einzelnen Bildschirmzeilen Punkte auf der Bildschirmoberfläche zum Leuchten. Die Intensität, d.h. die Helligkeit des Strahls, ist regelbar.

Der Strahl startet in der linken oberen Ecke und erzeugt in einer waagerechten Linie bis zum rechten Rand in der gewünschten Helligkeit die einzelnen Bildpunkte. Dabei wird eine sogenannte "Horizontal Scan Line", eine waagerechte Bildschirmzeile erzeugt. Auf jeder waagerechten Bildschirmzeile befinden sich 228 Color Clocks (Bildpunkte). Nach dem Erreichen des rechten Rands wird der Strahl ausgeschaltet und um eine waagerechte Bildschirmzeile nach unten und gleichzeitig wieder an den rechten Rand bewegt. Diese "Leerlaufzeit" wird als sogenannter "Horizontal Blank" bezeichnet. In diesem "Horizontal Blank" kann der ANTIC den "Display List-Interrupt" (DLI) erzeugen, doch dazu später mehr.

Dieser Vorgang des Erzeugens einer Bildschirmzeile und der Rückführung des Elektronenstrahls wiederholt sich insgesamt 312mal pro Bild (bei PAL-Geräten). Ist die rechte untere Ecke erreicht, wird wiederum der Elektronenstrahl abgeschaltet und in die linke obere Ecke zurückgeführt. Diesen Zeitraum der Rücksetzung des Elektronenstrahls in die linke obere Ecke bezeichnet man auch als "Vertical Blank". Auch in diesem Zeitraum kann der ANTIC einen eigenen Interrupt, den "Vertical Blank Interrupt" erzeugen. Somit ist ein Bildaufbau abgeschlossen.

Diesen Bildaufbau führt der ANTIC nun 50mal pro Sekunde durch.

Vom Benutzer kann jedoch nicht jeder Punkt auf dem Bildschirm angesprochen werden. Da viele Fernseher einen Teil des Bilds "abschneiden" (sogenanntes "Overscan"), werden unterhalb und oberhalb des Anzeigefeldes insgesamt 120 Leerzeilen vom Bildschirmtreiber und der Hardware eingefügt, sodass nur noch 312-120=192 vom Benutzer beeinflussbare Bildschirmzeilen übrig bleiben (die Bildschirmzeilen 32 bis 223). Der vom ANTIC darstellbare Bereich geht von Bildschirmzeile 8 bis 247 und ist damit 240 Bildschirmzeilen hoch. Auf PAL-Fernsehern ist dieser Bereich meist komplett sichtbar, lediglich die obersten und untersten 8 Bildschirmzeilen dieses Bereichs können nicht sichtbar sein. Die Nummer der

augenblicklich erzeugten Bildschirmzeile geteilt durch zwei ist übrigens im Register VCOUNT (54283; $D40B) zu finden. Auch die Breite des Bildes beträgt nicht, zumindest beim Einschalten, die maximal mögliche Zahl von 228 Farbpunkten. Standardmäßig ist vielmehr der Wert von 160 Bildpunkten eingestellt. Dies entspricht 40 Zeichen der Grafikstufe 0. Es gibt daneben noch zwei weitere Bildbreiten:

- Das breite Anzeigefeld mit 192 Bildpunkten, von denen jedoch je nach Fernseher nur ca. 174 sichtbar sind. Dies entspricht ca. 44 sichtbaren Zeichen der Grafikstufe 0.

- Das schmale Anzeigefeld mit 128 Bildpunkten. Dies entspricht 32 Zeichen der Grafikstufe 0.

Die Bildbreite ist über das Register DMACTL (54272; $D400) bzw. über das Schattenregister SDMCTL (559; $22F) veränderbar.

2.1.1 Display List

Der ANTIC ist nun ein Mikroprozessor, der über ein eigenes Programm, die sogenannte "Display List", verfügt. Diese Display List ist eine Aneinanderreihung von speziellen Programmbefehlen für den ANTIC. Sie gibt an, wie die 192 sichtbaren Bildschirmzeilen auszusehen haben. Grundsätzlich kennt der ANTIC drei Arten von Programmbefehlen:

Leerzeilen

Dieser Ein-Byte-Befehl wird dazu verwendet, eine bis acht leere Bildschirmzeilen in der Hintergrundfarbe zu erzeugen. Das verwendete Befehlsformat sieht dabei wie folgt aus:

Bit	Funktion
7	1=DLI wird ausgelöst. Nicht zusammen mit Bit 6 setzen!
6-4	0-7=1-8 Leerzeilen
3-0	Diese Bits müssen 0 sein.

Sprungbefehle

Der ANTIC hat wie die 6502-CPU einen 16 Bits breiten Programmzähler. Hier heißt er "Display List Counter". Durch die drei Bytes umfassenden Sprungbefehle wird er mit einem neuen Wert geladen, sodass Sprünge

innerhalb des Speichers möglich sind. Das zweite Byte eines Sprungbefehls ist das niederwertige, das dritte das höherwertige Byte der Sprungadresse. Da die oberen sechs Bits des Display-List-Counters fest sind und nur bei Sprungbefehlen neu geladen werden, kann die Display List keine 1 KByte-Grenze überschreiten, ohne dass ein Sprungbefehl verwendet wird! Der Display List-Counter wird durch Einschreiben der Anfangsadresse der neuen Display List in DLISTH/L (54274,54275; $D402,$D403) bzw. in die Schattenregister SDLSTL/H (560,561; $230, $231) auf den Beginn einer Display List initialisiert. Die Sprungbefehle:

Bit	Funktion
7	1=DLI wird ausgelöst.
6	0=direkter Sprung, der eine Leerzeile erzeugt.
	1=Sprung, bei dem auf das Ende des nächsten "Vertical Blank Interrupts" gewartet wird. Am Schluss einer Display List steht üblicherweise ein solcher Sprung zum Anfang der Display List.
5,4	Zustand der Bits ist nicht relevant
3-0	1=Sprungbefehl

Anzeigebefehle

Diese Befehlsart kann ein oder drei Bytes umfassen. Die ein Byte langen Befehle geben eine von vierzehn möglichen Grafikstufen an, in der die nächsten 1, 2, 4, 8, 10 oder 16 Bildschirmzeilen dargestellt werden sollen. Bei den drei Bytes langen LMS-Befehlen (LMS = "Load Mem Scan") wird ähnlich wie bei den Sprungbefehlen der sogenannte "Memory Scan Counter" auf einen neuen Wert gesetzt. Der Memory Scan Counter enthält die Adresse, von der ab der Bildschirmspeicher beginnt. In jeder Display List muss also mindestens ein LMS stehen, damit der Beginn des Bildspeichers definiert ist. Die oberen vier Bits des Memory Scan Counters werden nur beim LMS-Befehl verändert. Der Memory Scan Counter ist also ähnlich wie der Display List Counter nur zwölf Bits breit. Ein auf dem Bildschirm darzustellender Speicherbereich kann deshalb ohne LMS-Befehl keine 4-KByte-Grenze überschreiten! Daher ist es ggf. nötig, mehrere LMS-Befehle innerhalb einer Display List einzusetzen.

Außerdem kann durch Setzen von Bit 4 und 5 horizontales bzw. vertikales Feinscrolling ermöglicht werden. Feinscrolling bedeutet, dass Bildschirmteile in Zeichensatzgrafik nicht zeichenweise, sondern viel feiner um einzelne Bildpunkte bewegt werden. Eine genauere Erklärung des Feinscrollens erfolgt später.

Horizontales Scrolling		■		■		■		■		■		■		■		■
Vertikales Scrolling			■	■			■	■			■	■			■	■
Bildspeicheradr. laden					■	■	■	■					■	■	■	■
Display List Interrupt									■	■	■	■	■	■	■	■
1 Leerzeile	00								80							
2 Leerzeilen	10								90							
3 Leerzeilen	20								A0							
4 Leerzeilen	30								B0							
5 Leerzeilen	40								C0							
6 Leerzeilen	50								D0							
7 Leerzeilen	60								E0							
8 Leerzeilen	70								F0							
Sprung (Jump)	01								81							
Sprung & VBI	41								C1							
* Bedeutung der Klammer siehe unten																
Text (40,2,8)	02	12	22	32	42	52	62	72	82	92	A2	B2	C2	D2	E2	F2
Text (40,2,10)	03	13	23	33	43	53	63	73	83	93	A3	B3	C3	D3	E3	F3
Text (40,5,8)	04	14	24	34	44	54	64	74	84	94	A4	B4	C4	D4	E4	F4
Text (40,5,16)	05	15	25	35	45	55	65	75	85	95	A5	B5	C5	D5	E5	F5
Text (20,5,8)	06	16	26	36	46	56	66	76	86	96	A6	B6	C6	D6	E6	F6
Text (20,5,16)	07	17	27	37	47	57	67	77	87	97	A7	B7	C7	D7	E7	F7
Punkt (40,4,8)	08	18	28	38	48	58	68	78	88	98	A8	B8	C8	D8	E8	F8
Punkt (80,2,4)	09	19	29	39	49	59	69	79	89	99	A9	B9	C9	D9	E9	F9
Punkt (80,4,4)	0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
Punkt (160,2,2)	0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB
Punkt (160,2,1)	0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
Punkt (160,4,2)	0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD
Punkt (160,4,1)	0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
Punkt (320,2,1)	0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF

Tab. 2.1: ANTIC-Befehlsübersicht

* Anzahl Bytes pro Zeile/Anzahl Farben/Anzahl Bildschirmzeilen pro Zeichen bzw. Punkt. Zahlenangaben für die Anzeigebefehle sind in hexadezimaler Schreibweise.

Ein Anzeigebefehl hat folgendes Format:

Bit	Funktion
7	1=DLI wird ausgelöst.
6	0=Der Befehl umfasst nur ein Byte.
	1=Der Befehl umfasst drei Bytes, der "Memory Scan Counter" wird neu geladen.
5	1=Vertikales Feinscrolling ist möglich.
4	1=Horizontales Feinscrolling ist möglich.
3-0	2-15=Grafikmodus wird ausgewählt.

2.1.2 Grafikstufen

Folgende Grafikstufen werden vom ANTIC unterstützt:

Textgrafikstufen

Hier repräsentiert jedes Byte im Speicher ein einzelnes Zeichen auf dem Bildschirm. Dazu muss durch Einschreiben des höherwertigen Bytes der Adresse in CHBASE (54281; $D409) bzw. in das Schattenregister CHBAS (756; $2F4) die Anfangsadresse eines Zeichensatzes festgelegt werden. Normalerweise muss ein Zeichensatz auf einer 1 KByte-Grenze beginnen. Für die Definition jedes Zeichens sind acht Bytes vorhanden, die das grafische Erscheinungsbild steuern. Ein Beispiel für die Definition eines einzelnen Zeichens und die Errechnung der Werte für den Zeichensatz:

Bit										
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Werte für den Zeichensatz**		
									=	0
			■	■				16+8	=	24
		■	■	■	■			32+16+8+4	=	60
	■	■			■	■		64+32+4+2	=	102
	■	■			■	■		64+32+4+2	=	102
	■	■	■	■	■	■		64+32+16+8+4+2	=	126
	■	■			■	■		64+32+4+2	=	102
									=	0

Tab. 2.2: Beispiel für Zeichensatzdefinition

Die Definition der Zeichen im Zeichensatz erfolgt nicht in der Reihenfolge der ATASCII-Zeichen, sondern in der internen Reihenfolge, die in der Gesamttabelle im Tabellenteil aufgeführt ist.

Wie wird nun ein Zeichen auf den Bildschirm gebracht?

Grafikstufen mit 40 Zeichen pro Zeile

Das höchstwertige Bit jedes Bytes aus dem Bildspeicher gibt an, ob das Zeichen invertiert dargestellt werden soll oder nicht. Die übrigen sieben Bits geben das darzustellende Zeichen an. Es kann hierbei aus V = 128 Zeichen ausgewählt werden. Die Nummer des Zeichens wird mit acht (jede Zeichendefinition ist acht Bytes lang!) multipliziert, um die Adresse des Zeichens innerhalb des Zeichensatzes zu erhalten. Dazu wird die Anfangsadresse des Zeichensatzes addiert. Das High-Byte wird durch CHBASE angegeben, das Low-Byte ist immer 0. So ergibt sich also die Adresse der Daten des Zeichens innerhalb des Speichers. Die acht Byte lange Zeichenmatrix wird in den acht aufeinanderfolgenden Bildschirmzeilen untereinander auf den Bildschirm gebracht.

Grafikstufen mit 20 Zeichen pro Zeile

Hier geben die oberen zwei Bits das Farbregister (COLPF0-3: 53270-53273) an, in dessen Farbe das Zeichen dargestellt werden soll. Nur die unteren sechs Bits wählen daher eins aus 64 möglichen Zeichen aus. Der benötigte Zeichensatz ist daher auch nur 64 * 8 = 512 Bytes lang. Die weitere Zeichendarstellung verläuft wie oben.

Die Tabelle 2.3 zeigt die wichtigen Angaben zu den verschiedenen Textgrafikstufen.

Besonderheiten der einzelnen Textmodi

ANTIC-Modus 2: Dieser ANTIC-Modus entspricht der Grafikstufe 0. Er dürfte in der Anwendung den meisten BASIC-Benutzern hinreichend bekannt sein.

ANTIC-Modus 3: Dieser ANTIC-Modus entspricht in der Auflösung der einzelnen Zeichen dem ANTIC-Modus 2. Er ist jedoch zehn Bildschirmzeilen hoch, zwei mehr als normal. Dies erlaubt die Definition von Zeichen mit echten Unterlängen. Das letzte Viertel des Zeichensatzes wird dazu um zwei Bildschirmzeilen tiefer als der übrige Zeichensatz dargestellt. Bei den ersten drei Vierteln des Zeichensatzes werden zwei leere Bildschirmzeilen unten angefügt, bei Zeichen aus dem letztem Viertel (also auch den Kleinbuchstaben) werden diese zwei Zeilen oberhalb dargestellt und

die ersten beiden Bytes des Zeichens in den untersten beiden Bildschirm-zeilen angezeigt (siehe Abb. 2.1).

ANTIC-Modus	**Grafikstufe**	**Zahl der Farben**	**Zeichen pro Zeile**	**Bildschirmzeilen pro Zeichen**	**Color Clocks pro Pixel**	**Bits pro Pixel**	**Bits in den Daten**	**Bits in den Zeichen**	**Farbregister**
2	0	1,5	40	8	0,5	1	0 1	— —	PF2 PF1 (Helligkeit)
3	—	1,5	40	10	0,5	1	0 1	— —	PF2 PF1 (Helligkeit)
4	12 bei XL/XE	5	40	8	1	2	00 01 10 11 11	Bit7 =0 Bit7 =1	BAK PF0 PF1 PF2 PF3
5	13 bei XL/XE	5	40	16	1	2	00 01 10 11 11	Bit7 =0 Bit7 =1	BAK PF0 PF1 PF2 PF3
6	1	5	20	8	1	1	— 00 01 10 11	0 1 1 1 1	BAK PF0 PF1 PF2 PF3
7	2	5	20	16	1	1	— 00 01 10 11	0 1 1 1 1	BAK PF0 PF1 PF2 PF3

Tab. 2.3: Übersicht der Textgrafikstufen

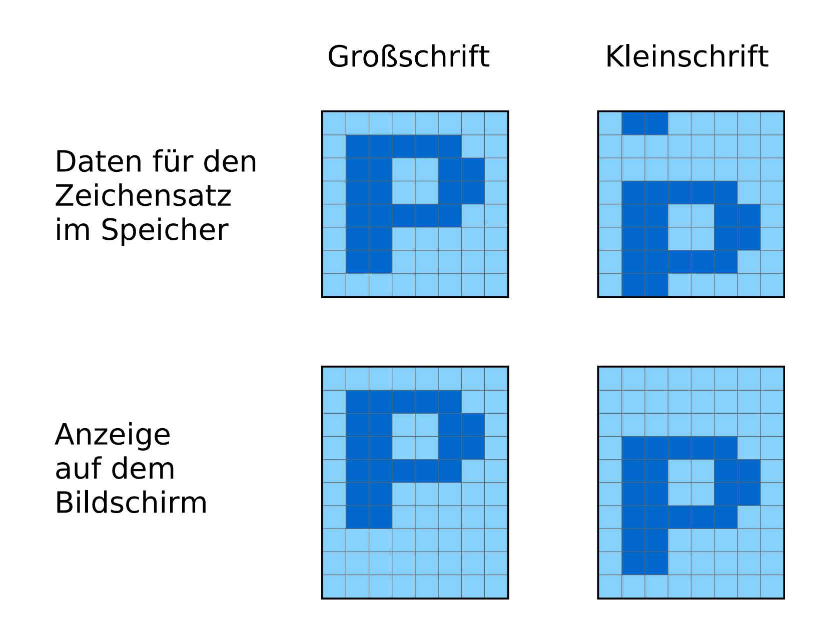


Abb. 2.1: Beispiel für die Anwendung des ANTIC-Modus 3

ANTIC-Modus 4: Diesem ANTIC-Modus entspricht bei den XL/XE-Geräten die Grafikstufe 12. Die Zeichen sind hierbei genauso groß wie in der Grafikstufe 0, jedoch besteht jedes Zeichen horizontal aus vier Pixel statt acht. Da für jedes Pixel nun zwei Bits zur Verfügung stehen, wird eins von vier möglichen Farbregistern ausgewählt.

Dies erlaubt die mehrfarbige Darstellung von Landschaften, Karten, Gebäuden usw. Eine fünfte Farbe kann durch Setzen von Bit 7 (Invertierung des Zeichens) folgendermaßen angesprochen werden: Taucht in diesem Fall in der Definition des Zeichens die Bitkombination 11 auf, so wird dann die Farbe aus dem Farbregister COLOR3 (711; $2C7) und nicht aus Register COLOR2 (710; $2C6) übernommen. Insgesamt können also (ohne DLI) fünf Farben gleichzeitig dargestellt werden.

Dieser Grafikmodus wird in Verbindung mit horizontalem und vertikalem Fine-Scrolling sehr häufig zur Darstellung des Bildhintergrundes bei Spielen verwendet. Als Beispiel sei hier nur CAVELORD (AXIS Komputerkunst) genannt (siehe Abb. 2.2 und 2.3).



Abb. 2.2: CAVELORD (AXIS Komputerkunst)



Abb. 2.3: CAVELORD (AXIS Komputerkunst)

Durch die Benutzung verschiedener Helligkeiten einer Farbe können mit ANTIC-Modus 4 auch interessante dreidimensionale Effekte verwirklicht werden (siehe Abb. 2.4).

ANTIC-Modus 5: Diesem ANTIC-Modus entspricht bei den XL/XE-Geräten die Grafikstufe 13. Die Darstellung der Zeichen sieht genauso aus wie beim ANTIC-Modus 4, jedoch ist nun jedes Pixel zwei Bildschirmzeilen, also doppelt so hoch.

ANTIC-Modus 6: Dieser ANTIC-Modus entspricht der Grafikstufe 1. Die obersten zwei Bits des darzustellenden Zeichens wählen ein Farbregister aus, die unteren sechs Bits wählen das Zeichen im Zeichensatz an. In diesem und im folgenden ANTIC-Modus ist der Zeichensatz deshalb auch nur 2^6=64 Zeichen lang. Da jedes Pixel doppelt so breit ist wie im ANTIC-Modus 2, umfasst ein Zeichen des ANTIC-Modus 6 zwei Zeichen des ANTIC-Modus 2; es werden also horizontal nur 20 Zeichen dargestellt.

ANTIC-Modus 7: Dieser ANTIC-Modus entspricht der Grafikstufe 2. Der Aufbau der Zeichen ist vergleichbar dem des ANTIC-Modus 6, jedoch ist jedes Zeichen sechzehn Bildschirmzeilen hoch. Das einzelne Zeichen ist somit doppelt so hoch und breit wie ein normales Zeichen.



Abb. 2.4: Highway Duel

Zusätzlich kann die Darstellungsweise der einzelnen Zeichen durch das Register CHACTL (54273; $D401, Schattenregister CHACT: 755; $2F3) beeinflusst werden. Von einer Änderung auf dem Bildschirm sind jedoch nur die Zeichen betroffen, deren Bit 7 im Bildschirmspeicher gesetzt ist, die also in der Grafikstufe 0 invers dargestellt werden.

Dabei ist es möglich, die Zeichen auf den Kopf zu stellen, sie verschwinden zu lassen usw. Eine genaue Beschreibung dieses Registers ist im Speicherplan bei CHACT zu finden.

Bei den Textgrafikstufen ist weiterhin das sogenannte "Fine Scrolling" möglich. Dabei werden die einzelnen Zeichen nicht nur zeichenweise, sondern sogar pixelweise verschoben. Wie das horizontale und vertikale Feinscrolling programmiert wird, wird in einem eigenen Abschnitt dieses Kapitels erklärt.

Punktgrafikstufen

In diesen Punktgrafikstufen repräsentiert jedes Byte aus dem Bildspeicher eine bestimmte Zahl von Punkten auf dem Bildschirm. Die Daten im Bildspeicher sind zeilenweise angeordnet.

Dabei sind grundsätzlich zwei verschiedene Darstellungsweisen zu unterscheiden.

Grafikstufen mit vierfarbiger Darstellung

Hier ist jedes Pixel zwei Bits breit. Diese zwei Bits erlauben die Auswahl einer Farbe von vier möglichen. Die Größe der Pixel liegt zwischen ein und vier Farbpunkten ("color clocks") in der Breite und zwischen einer und acht Bildschirmzeilen in der Höhe. Grafikstufen, bei denen die einzelnen Pixel besonders groß sind, verbrauchen natürlich wesentlich weniger Speicherplatz als die extrem hochauflösenden Grafikstufen.

Grafikstufen mit zweifarbiger Darstellung

Hier ist jedes Pixel nur ein Bit breit. Ist das Bit gesetzt, wird die Farbe aus dem ersten Farbregister (COLPF0: 53270; $D016) gewählt, ist es nicht gesetzt, wird dieser Pixel in der Hintergrundfarbe dargestellt. In jedem Byte der Bildspeicherdaten werden also acht Pixel (Bildpunkte) abgespeichert.

Eine Ausnahme innerhalb dieser Grafikstufen bildet der ANTIC-Modus 15 (Grafikstufe 8). Da die horizontale Auflösung über die des Fernsehers/Monitors hinausgeht, können hier nur Helligkeiten eingestellt werden. Ist ein Bit gesetzt, so nimmt dieses Pixel die Helligkeit vom Farbregister COLPF1 (53271; $D017) an, ansonsten wird die Helligkeit des Farbregisters COLPF2 (53272; $D018) gewählt. Der Helligkeitswert in einem Farbregister wird in den unteren vier Bits abgespeichert. Die Farbe des Bildschirms kann im Farbregister COLPF2 eingestellt werden.

Tabelle 2.4 enthält die Angaben zu den verschiedenen Punktgrafikstufen.

ANTIC-Modus	Grafik-stufe	Zahl der Farben	Punkte pro Zeile	Bytes pro Bild-schirm-zeile	Color Clocks pro Pixel	Bild-schirm zeilen pro Pixel	Bits pro Pix-el	Bits im Pix el	Farb-regi-ster
8	3	4	40	10	4	8	2	00 01 10 11	BAK PF0 PF1 PF2
9	4	2	80	10	2	4	1	0 1	BAK PF0
10	5	4	80	20	2	4	2	00 01 10 11	BAK PF0 PF1 PF2
11	6	2	160	20	1	2	1	0 1	BAK PF0
12	14 bei XL/ XE	2	160	20	1	1	1	0 1	BAK PF0
13	7	4	160	40	1	2	2	00 01 10 11	BAK PF0 PF1 PF2
14	15 bei XL/ XE	4	160	40	1	1	2	00 01 10 11	BAK PF0 PF1 PF2
15	8	1,5	320	40	0,5	1	1	0 1	PF2 PF1 (Hel-lig-keit)

Tab. 2.4: Übersicht der Punktgrafikstufen

2.1.3 Der Aufbau der Display List

Wie ist nun eine normale Display List der Grafikstufe 0 aufgebaut? Im Falle eines ATARI-Computers mit 48 KByte Speicher sieht die Aufteilung wie folgt aus:

```
Adresse   Daten im Speicher

$BC20     $70
          $70   3*8= 24 Leerzeilen (für Fernseher mit Overscan)
          $70
          $42   ANTIC-Modus 2 mit LMS-Befehl
          $40   Der Memory Scan Counter wird auf den folgenden
          $BC   Bildschirmspeicherbereich gesetzt.
          $02
          $02
          $02
          $02
          .
          .     23 weitere Zeilen im ANTIC-Modus 2
          .
          $02
          $02
          $02
          $02
          $41   Sprung zum Anfang der Display List,
          $20   gleichzeitig wird auf den VBI gewartet.
          $BC

$BC40     24*40 = 960 Bytes für den Bildspeicher
```

Zahlenangaben für Adressen und Anzeigebefehle sind in hexadezimaler Schreibweise!

Diese Display List ist extrem einfach aufgebaut, da sie nur eine Grafikstufe enthält und nur einen LMS-Befehl. Um diese Display List einzuschalten, muss in die Register DLISTL/H (54274,54275; $D402, $D403) die Startadresse der Display List geschrieben werden. In diesem Fall wäre es die Adresse $BC20. Nicht jede Display List muss so einfach aufgebaut sein. Grafikstufen können beliebig gemischt werden, verschiedene Speicherbereiche können auf den Bildschirm gebracht werden usw. Ein Beispiel für eine kompliziertere Display List ist zum Beispiel der Screenshot des Vorspanns "AXIS Komputerkunst" in Abb. 2.5. Hier werden verschiedene Grafikstufen gemischt.



Abb. 2.5: Vorspann AXIS Komputerkunst

Eine weitere Möglichkeit, den Bildschirmaufbau zu beeinflussen, wird im folgenden Abschnitt vorgestellt:

2.1.4 Der Display List Interrupt

Der Display List Interrupt (DLI) kann dazu verwendet werden, zwischen verschiedenen Farben, Zeichensätzen usw. während des Bildschirmaufbaus umzuschalten. Damit ein DLI ausgelöst wird, müssen verschiedene Voraussetzungen gegeben sein. Zunächst muss an der Stelle in der Display List, an der der DLI erfolgen soll, beim entsprechenden ANTIC-Befehl das höchstwertigste (das siebte) Bit gesetzt sein. Als Nächstes muss die DLI-Routine an eine vom Anwender zu bestimmende Stelle im Speicher gelegt werden. Diese Routine muss nur die Hardwareregister für Farben usw. ändern, da sich eine Änderung des Schattenregisters erst beim Durchlaufen des VBI (also nach Beendigung des Bildschirmaufbaus) bemerkbar macht. Außerdem hätte eine Veränderung des Schattenregisters die Folge, dass der Ursprungswert durch den VBI nicht mehr in das Hardwareregister geschrieben wird. Ein Beispiel für eine solche Routine folgt nachstehend. Zusätzlich sollte vor jeder Änderung eines Farbregisters ein "STA WSYNC" stehen, um die horizontale Synchronisation zu gewährleisten. Ansonsten kann es zum Flackern der Farben in der DLI-Bildschirmzeile kommen. Anschließend wird die Adresse der DLI-Routine in das Register VSDLST (512,513; $200, $201) geschrieben. Da es sich beim DLI um einen nicht maskierbaren Interrupt handelt, muss in dem Register

NMIEN (54286; $D40E), das die nicht maskierbaren Interrupts ermöglicht, ebenfalls das höchstwertige Bit gesetzt werden. Durch das OS sind nur VBIs vorgesehen, so muss auf jeden Fall NMIEN geändert werden (normalerweise auf 192 ($C0)).

Nachstehend ein Beispiel für eine DLI-Routine:

```
DLI PHA            ;Alle benötigten Prozessorregister
                   ;müssen auf den Stapel "gerettet" werden.
    STA WSYNC      ;Warten auf horizontale Synchronisation.
    LDA #$4A       ;der neue Farbwert muss nur in das
    STA COLPF0     ;Hardwareregister geschrieben werden.
    LDA #>NEWFONT  ;Der Zeichensatz wird
    STA CHBASE     ;ebenfalls umgeschaltet.
    PLA            ;Alle Register müssen nun zurückgeholt
                   ;werden.
    RTI            ;Rückkehr ins Hauptprogramm
```

Die entsprechende Initialisierungsroutine sieht wie folgt aus (unter der Voraussetzung, dass bereits eine Display List eingeschaltet ist, die einen ANTIC-Befehl enthält, dessen höchstwertiges Bit gesetzt ist):

```
    LDA #<DLI
    STA VSDLST
    LDA #>DLI    ;Zuerst wird die Adresse der
    STA VSDLST+1 ;DLI-Routine gesetzt.
    LDA #$C0
    STA NMIEN    ;Jetzt erst gelangt der DLI zur
                 ; Ausführung.
```

Die Programmierung eines DLI ist also insofern nicht ganz einfach, als dass jeder einzelne Schritt vom Anwender selbst durchgeführt werden muss.

Ein Beispiel für die Anwendung von Display List-Interrupts ist der Screenshot von "Memo-Box", bei dem die unterschiedlich hellen waagerechten "Streifen" per DLI erzeugt werden.

Die zweite, vom Bildschirmaufbau abhängende Interruptart wird im nachstehenden Abschnitt vorgestellt.

2.1.5 Der Vertical-Blank-Interrupt (VBI)

Dieser Interrupt wird in dem Moment ausgelöst, in dem der Elektronenstrahl, der das Fernsehbild erzeugt, am Ende des Bildes (rechts unten) angelangt ist. Der Elektronenstrahl muss nun (unsichtbar!) nach links oben zurückbewegt werden. In dieser für einen Computer sehr langen Zeit kann der VBI laufen. Da während des VBI garantiert kein Bild aufgebaut wird, ist der VBI sehr gut geeignet, Änderungen des Bildes durchzuführen. Es kann, da der Elektronenstrahl ausgeschaltet ist, nicht zu irgendwelchen Bildstörungen (Flackern etc.) kommen. Ein VBI wird jedoch



Abb. 2.6: Memo-Box

dann nicht ausgelöst, wenn durch das Programm Bit 6 von NMIEN (54286; $D40E) zurückgesetzt wird. Durch das OS wird dieses Bit beim Einschalten gesetzt, will man jedoch die Auslösung eines VBI verhindern, so muss man dieses Bit zurücksetzen. Die Verwaltung der Schattenregister, das Verringern der Systemzähler, die Tastenwiederholfunktion wird im sogenannten System-VBI durchgeführt. Die Struktur des gesamten VBI sieht wie folgt aus:

Auslösung eines NMI
↓
Test durch die OS-NMI-Routine
↓

Sprung durch VVBLKI (546,547; $222,$223)
↓
↓ → mögliche 'Immediate' VBI-Routine des Anwenders
↓
JMP SYSVBV ($E45F)
↓
System-VBI-Routine des OS
↓
Sprung durch VVBLKD (548,549; $224,$225)
↓
↓ → mögliche 'Deferred' VBI-Routine des Anwenders
↓
JMP XITVBV ($E462)
↓
Rückkehr in das Hauptprogramm

Im Wesentlichen sind durch den Anwender also zwei verschiedene Arten des VBI programmierbar:

Der "Immediate" VBI

Er wird in jedem Fall ausgeführt, auch bei zeitkritischen I/O-Operationen. Die Anfangsadresse der neuen "Immediate" VBI-Routine muss in den Vektor VVBLKI (546,547; $222,$223) geschrieben werden. Die Veränderung dieses Vektors geschieht am besten durch Verwendung der SETVBV-Routine (58460; $E45C). Ein beispielhafter Einsprung in diese Routine sieht wie folgt aus:

```
LDA #6        ;6='Immediate' VBI
              ;7='Deferred' VBI
LDX #>VBROUT ;Das X-Register muss das High-Byte der
              ;Adresse der neuen VBI-Routine enthalten.
LDY #<VBROUT ;Das Y-Register enthält das Low-Byte.
JSR SETVBV    ;nun wird die neue Adresse gesetzt.
```

Das Benutzen der SETVBV-Routine hat den Vorteil, dass es nicht zu unerwünschten Abstürzen kommen kann, weil in dem Moment, in dem erst ein Byte des Vektors verändert ist, ausgerechnet ein VBI ausgelöst wird. In diesem Fall zeigt der Vektor auf eine nicht sehr sinnvolle Adresse; ein Absturz des Computers ist dann meist die Folge.

Soll der normale System-VBI noch ausgeführt werden, so muss am Ende der neuen VBI-Routine ein JMP SYSVBV (=JMP $E45F) stehen. Der "Immediate VBI" sollte nicht mehr als 4500 Maschinenzyklen umfassen.

Der "Deferred" VBI

Dieser VBI wird nach der System-VBI-Routine ausgeführt, jedoch nur dann, wenn in diesem Augenblick keine zeitkritische I/O-Operation stattfindet. Jede I/O-Operation, die über die → SIO läuft, setzt das CRITIC-Flag (66; $42) auf einen Wert ungleich von 0. Die System-VBI-Routine testet nun, ob dieses CRITIC-Flag gesetzt ist. Ist das der Fall, so wird die System-VBI-Routine abgebrochen und in das Hauptprogramm zurückgekehrt, um so viel Zeit wie möglich für die I/O-Operation zur Verfügung zu stellen. Der "Deferred" VBI kann etwa 24000 Maschinenzyklen umfassen, ohne dass Probleme auftreten.

Eine kurze Übersicht über das, was in der System-VBI-Routine geschieht:

- RTCLOK (18,19,20; $12,$13,$14) wird um eins erhöht.

- Der ATRACT-Timer (77; $4D) wird um eins erhöht, falls er größer als 127 ($7F) wird, wird DRKMSK (78; $4E) auf 246 ($F6) (normal ist der Wert 254 ($FE)) und COLRSH (79; $4F) gesetzt. Für COLRSH ist der normale Wert 0. DRKMSK und COLRSH werden später noch verwendet.

- Der Systemzähler 1 wird um eins verringert (wenn er nicht 0 ist), und falls er 0 erreicht, wird indirekt durch CDTMA1 (550,551; $226,$227) gesprungen.

- Wenn nun entweder das CRITIC-Flag oder das Interrupt-Flag des Prozessors gesetzt ist, wird der VBI beendet. Die folgenden Teile des System-VBIs werden also nur dann ausgeführt, wenn keine I/O-Operation stattfindet.

- Nun werden einige Schattenregister in die zugehörigen Hardwareregister übertragen.

Diese Register sind:

Schattenregister	→ **Hardwareregister**
SDLSTL/H ($230,$231)	→ DLISTL/H ($D402,$D403)
SDMCTL ($22F)	→ DMACTL ($D400)
GPRIOR ($26F)	→ PRIOR ($D01B)
PCOLR0-3 ($2C0-$2C3)	→ COLPM0-3 ($D012-$D015)
COLOR0-4 ($2C4-$2C8)	→ COLPF0-3, COLBK ($D016-$D01A)
CHBAS ($2F4)	→ CHBASE ($D409)
CHACT ($2F3)	→ CHACTL ($D401)

Folgende Hardwareregister werden in die entsprechenden Schattenregister übertragen:

Hardwareregister	**→ Schattenregister**
PENV ($D40D)	→ LPENV ($235)
PENH ($D40C)	→ LPENH ($234)
PORTA/B ($D300,$D301)	→ STICK0-7 ($278-$27B),
	→ PTRIG0-7 ($27C-$283)
TRIG0-3 ($D010-$D013)	→ STRIG0-3 ($284-$287)
POT0-7 ($D200-$D207)	→ PADDL0-7 ($270-$277)

− Die Farbregister werden bei der Übertragung zunächst mit COLRSH exclusiv oderiert und anschließend mit DRKMSK undiert. Dies macht sich normalerweise nicht bemerkbar. Wird jedoch ca. 11 Minuten keine Taste betätigt, so erreicht der Zähler ATRACT einen Wert, der größer als 127 ist. Dann werden COLRSH und DRKMSK so gesetzt, dass die Helligkeit der Farben reduziert wird, um ein Einbrennen des Fernsehbildes zu verhindern.

− Die Systemzähler 2 bis 5 werden dekrementiert (wenn sie nicht 0 sind), und falls einer 0 erreicht, wird durch den zugehörigen Vektor gesprungen bzw. das zugehörige Flag gesetzt.

− Schließlich wird noch die Tastenwiederholung durchgeführt.

Bei den XL/XE-Geräten wurde der System-VBI wie folgt erweitert:

− Das Register TRIG3 (53267; $D013) wird mit dem Schattenregister GINTLK (1013; $3FA) verglichen. TRIG3 enthält den Wert eins, wenn ein Programmmodul eingesteckt ist, andernfalls den Wert 0. Unterscheiden sich TRIG3 und GINTLK, so ist ein Modul eingesteckt oder herausgenommen worden. In diesem Fall bleibt der Computer, um einen Absturz zu verhindern, in einer endlosen Schleife.

− Ist das Register FINE (622; $26E) auf einen Wert ungleich von 0 gesetzt, als der Bildschirm neu geöffnet wurde, so scrollt von nun an der Bildschirm fein. Dieses wird ebenfalls im System-VBI durchgeführt.

2.1.6 Direct Memory Access

Um überhaupt ein Fernsehbild erzeugen zu können, muss der ANTIC "direkt auf den Programmspeicher zugreifen" können. Dazu schaltet er in bestimmten Zeitabständen die 6502-CPU des Computers aus und greift eigenständig auf den Speicher zu. Dieser "Direct Memory Access" (DMA) findet z.B. bei der Zeichen- oder Playerdarstellung eine entscheidende

Anwendung. Wie der DMA technisch funktioniert, ist für den Anwender (Programmierer) nicht ganz so wichtig. Er muss nur wissen, dass durch DMA der Computer um ca. 20-30% langsamer wird. Dies ist natürlich von der gewählten Grafikstufe abhängig.

Der DMA wird über das Register DMACTL (54272; $D400) bzw. über das zugehörige Schattenregister SDMCTL (559; $22F) kontrolliert. Es lässt sich genau festlegen, welche Art des DMA nun aktiviert wird. Die Bedeutung der einzelnen Bits dieses Registers ist an entsprechender Stelle im Speicherplan zu finden. Tipps zum Beschleunigen von Programmen:

- Ein schmaler Bildschirm verbraucht weniger Rechenzeit als ein breiter.

- Eine kurze Display List spart Rechenzeit gegenüber einer langen.

- Kurzzeitig kann sogar der gesamte Bildschirm ausgeschaltet werden. Um dem Anwender zu zeigen, dass der Rechner nicht abgestürzt ist, kann man das Hintergrundfarbregister kontinuierlich ändern.

- Auch die P/M-Grafik verbraucht Rechenzeit! Falls auf sie verzichtet werden kann, so ist es günstig, sie abzuschalten.

Für den direkten Speicherzugriff der Player/Missile-Grafik ist ein eigenes Hardwareregister zuständig (PMBASE: 54279; $D407). Es legt die Anfangsadresse des Speicherbereichs fest, der für die Player/Missile-Grafik reserviert werden soll. Die Programmierung von Playern und Missiles wird in einem eigenen Abschnitt ausführlich erläutert.

Der ANTIC verfügt ebenfalls über ein eigenes Register zur Festlegung des Beginns des Zeichensatzes (CHBASE: 54281; $D409, Schattenregister CHBAS: 756; $2F4). Ein Zeichensatz muss an einer 1-KByte-Grenze, bei den Grafikstufen 1 und 2 an einer 512-Byte-Grenze beginnen, damit der ANTIC sich die Daten korrekt holen kann.

2.1.7 Fine Scrolling

Mit dem Begriff "Fine Scrolling" ist ein pixelweises Verschieben von Zeichen in Textgrafikstufen gemeint. Texte, Skizzen usw. können so ohne störendes Ruckeln und Zucken über den Bildschirm bewegt werden. Es gibt im Wesentlichen zwei Arten des feinen Scrollings: horizontale und vertikale Bewegungen von Zeichen. Beide sind nicht ganz einfach zu programmieren. Eine Routine zum Feinscrollen wird am günstigsten in den VBI platziert, damit es während der Zeichenverschiebung nicht zu Bildstörungen durch den gleichzeitig ablaufenden Bildaufbau kommt.

Horizontales Fine Scrolling

Damit Zeichen in horizontaler Richtung fein bewegt werden können, muss in der zugehörigen ANTIC-Befehlszeile das Bit 4 des Befehls gesetzt sein. Ferner muss beachtet werden, dass sich der ANTIC beim Einschalten der Fine Scrollings in horizontaler Richtung mehr Zeichen als in normalen Bildzeilen holt. Ist die Bildbreite auf normal (40 Zeichen/ Zeile) geschaltet, werden 48 Zeichen dargestellt, ist der Bildschirm auf eng (32 Zeichen/Zeile) geschaltet, werden 40 Zeichen geholt. Dies geschieht deshalb, um ausreichend viele Zeichen pro Zeile darstellen zu können. Ist nämlich im normalen Modus die Zeile um ein halbes Zeichen nach links gescrollt, so sind (zumindest teilweise) 41 Zeichen auf dem Bildschirm zu sehen. Die Zahl der Farbpunkte, um die die einzelnen Zeichen nach rechts verschoben werden sollen, wird in HSCROL (54276; $D404) gespeichert. Ist das Zeichen um eine Zeichenbreite nach rechts (oder links) verschoben, muss die Anfangsadresse der entsprechenden Zeile um eins verringert (oder erhöht) werden. Ein Beispiel für eine Maschinensprachroutine, die innerhalb des VBI eine einzelne Zeile von links nach rechts scrollt, kann wie folgt aussehen:

```
       ...
 3000     DEC ZAEHLER         ;Zähler für das 'Fine
                              ;Scrolling',
 3010     LDA ZAEHLER         ;der alle 1/50 Sekunde (VBI-
                              ;Rhythmus)
 3020     AND #3              ;erhöht wird
 3030     STA HSCROL
 3040     EOR #3
 3050     BNE NICHTERHOEHEN
 3060     INC DLADR           ;Anfangsadresse der
 3070     BNE NOHIBYTE        ;Bildspeicherdaten
 3080     INC DLADR+1         ;dieser Zeile
 3090 NOHIBYTE
 3100     LDA DLADR
 3110     CMP #<ENDEZEILE     ;Ende der Zeile erreicht?
 3120     BNE NICHTERHOEHEN
 3130     LDA DLADR+1
 3140     CMP #>ENDEZEILE
 3150     BNE NICHTERHOEHEN
 3160     LDA #<ZEILENANFANG ;Wenn ja,DLADR
 3170     STA DLADR           ;zurücksetzen!
 3180     LDA #>ZEILENANFANG
 3190     STA DLADR+1
 3200 NICHTERHOEHEN           ;Ende der 'Fine Scroll' Routine
```

Wichtig zu beachten ist, dass die 3 in den Zeilen 3020 und 3040 der Anzahl der Pixel (-1) pro Zeichen in der jeweiligen Grafikstufe entspricht. Für Grafikstufe 1 oder 2 müsste dieser Wert auf 7 geändert werden.

Vertikales Fine Scrolling

Vertikales Scrolling wird im Prinzip ähnlich wie horizontales Scrolling durchgeführt. Durch Erhöhung des VSCROL-Registers (54277; $D405) werden diejenigen Zeilen bildschirmzeilenweise nach oben verschoben, deren Bit 5 des entsprechenden ANTIC-Befehles gesetzt ist. Die Skizze in Abb. 2.7 macht dies deutlich.

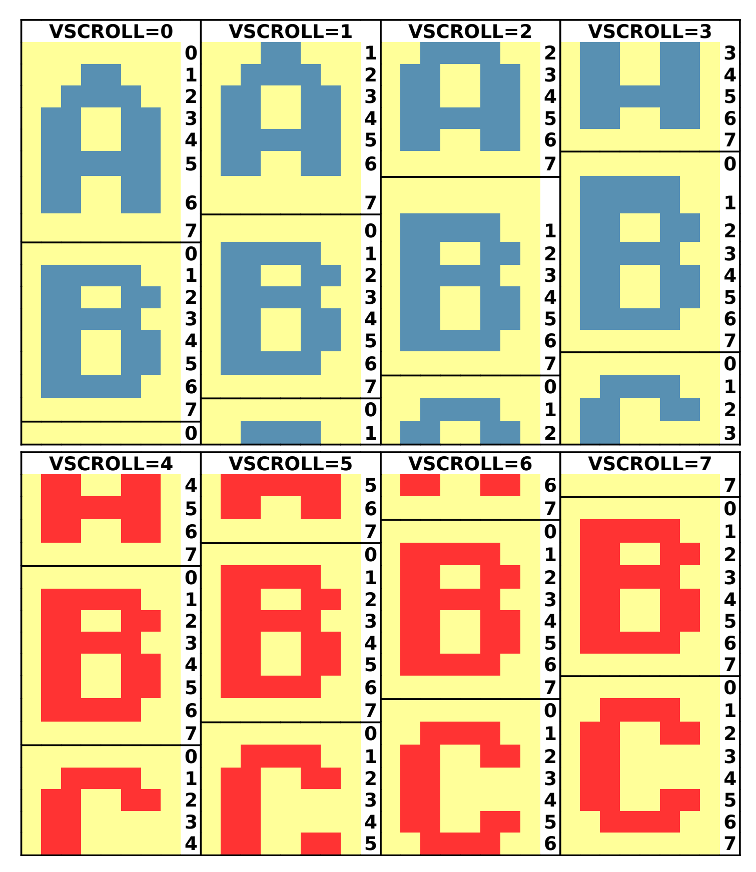


Abb. 2.7: Beispielskizze zum vertikalen Fine Scrolling

Nachdem VSCROL die Werte 0 bis 7 (bei Grafikstufe 0) durchlaufen hat, wird im LMS-Befehl für die entsprechende Bildschirmzeile die Adresse der Bildspeicherdaten um die jeweilige Bildschirmbreite erhöht (normalerweise um den Wert 40).

Sollen mehrere Zeilen übereinander fein gescrollt werden, so muss im Display List Befehl der letzten Zeile das Bit 5 nicht gesetzt sein! Wird Bit 5 der letzten zu scrollenden Zeile gesetzt, so erscheint z. B. bei einer gleichmäßigen Aufwärtsbewegung die letzte Zeile auf einen Schlag (sie wird genauso gescrollt wie die vorhergehenden Zeilen). Da dies aber nicht erwünscht ist, muss das vertikale feine Scrolling in der letzten Zeile nicht eingeschaltet werden.

Sollen mehrere übereinander liegende Zeilen sowohl horizontal als auch vertikal gescrollt werden (um zum Beispiel einen Ausschnitt einer Karte auf den Bildschirm zu bringen), so muss vor jeder Zeile ein neuer LMS-Befehl stehen. Alle LMS-Adressen müssen dann immer in gleicher Weise geändert werden.

Mithilfe von Display-List-Interrupts können natürlich auch die Fine-Scroll-Register verändert werden. Daher ist ohne Weiteres die Darstellung zweier unabhängig voneinander scrollender Bildfenster möglich (Abb. 2.8).



Abb. 2.8: Highway Duel

2.2 GTIA-Grafikstufen

Mit den drei nachstehend beschriebenen GTIA-Grafikstufen können die Signale, die der ANTIC produziert, noch weiter beeinflusst werden. Die drei Grafikstufen (GRAPHICS 9-11) haben eine Display List wie der ANTIC-Modus 15 (Grafikstufe 8), jedoch ist im Register PRIOR (53275; $D01B) bzw. im Schattenregister GPRIOR (623; $26F) eines der oberen zwei Bits gesetzt. Dadurch werden die Daten vom ANTIC anders als normal verarbeitet. Ein einzelnes Pixel ist nun statt einem halben Farbpunkt zwei Farbpunkte breit. Damit ergeben sich 2^4=16 Kombinationsmöglichkeiten, die in den einzelnen GTIA-Stufen unterschiedlich interpretiert werden. Die Pixel sind nun aber viermal so breit wie hoch. Für Grafikspezialisten ergeben sich dadurch vielfältige Möglichkeiten. Die einzelnen Grafikstufen:

2.2.1 GTIA-Modus 1 (Grafikstufe 9)

In dieser Farbstufe kann eine Farbe in 16 Helligkeiten dargestellt werden. Der Farbwert wird im Farbregister COLBK (53274; $D01A) bzw. im Schattenregister COLOR4 (712; $2C8) abgespeichert. Dieser GTIA-Modus ist für Schwarz-Weiß-Darstellungen in vielen Graustufen besonders geeignet. Digitalisierte Bilder wie das von Albert Einstein können besonders gut in diesem Modus dargestellt werden.

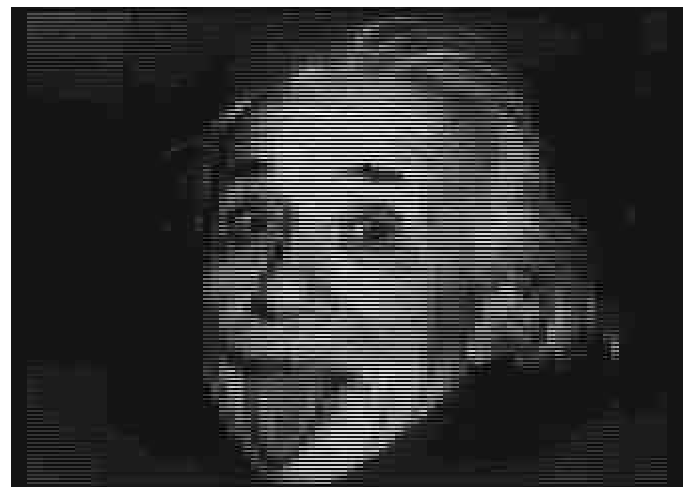


Abb. 2.9: Albert Einstein (digitalisiertes Foto)

2.2.2 GTIA-Modus 2 (Grafikstufe 10)

Hier stehen 9 Farben und Helligkeiten zur Verfügung. Die Zahl 9 kommt deshalb zustande, da der ATARI-Computer "nur" über neun Farbregister verfügt. Die gewünschten Farben werden in den Farbregistern COLPM0-COLBK (53266-53274; $D012-$D01A) bzw. in deren Schattenregistern PCOLR0-COLOR4 (704-712; $2C0-$2C8) abgespeichert. Hierbei sind Pixelwerte zwischen 0 und 8 möglich, wodurch eins der neun Farbregister angewählt wird. Die Hintergrundfarbe wird hier deshalb ausnahmsweise im Register COLPM0 (53266; $D012) bzw. im Schattenregister PCOLR0 (704; $2C0) abgespeichert. Dieser Modus hat den Vorteil, möglichst viele frei definierbare Farben gleichzeitig auf den Bildschirm zu bringen.

2.2.3 GTIA-Modus 3 (Grafikstufe 11)

Hier können 16 Farben in einer Helligkeit dargestellt werden. Der Helligkeitswert wird im Register COLBK (53274; $D01A) bzw. im Schattenregister COLOR4 (712; $2C8) gespeichert. Kombiniert man diesen Modus mit der ersten GTIA-Grafikstufe, indem abwechselnd per DLI zwischen dem ersten und dem dritten Modus umgeschaltet wird, so kann in einer relativ groben Auflösung ein Farbbild in allen 256 Farben dargestellt werden.

Die GTIA-Modi lassen sich nicht nur mit Grafikstufe 8 verwenden, auch die Kombination mit anderen ANTIC-Stufen ist denkbar. Diese Kombination wird durch Öffnen einer neuen Grafikstufe und Änderung des PRIOR-Registers erzielt. Die dabei entstehenden Effekte sind ähnlich wie bei den drei GTIA-Grafikstufen. Ob sich dabei jedoch sinnvolle Grafiken ergeben, hängt stark vom jeweiligen Programm ab.

2.3 ASCII/ATASCII

ASCII ist die Abkürzung für "American Standard Code for Information Interchange". Der ASCII-Code beinhaltet die Codes von 0 bis 127, wobei es sich bei den ersten 32 Bytes um Steuerbytes und Sonderzeichen handelt. ATASCII bedeutet einfach "ATari ASCII". Der ATASCII-Code entspricht für die Werte zwischen 32 und 127 im Wesentlichen dem ASCII-Code, während die ersten 32 Zeichen die ATARI-spezifischen Sonderzeichen enthalten (→ Tabellenteil). Die Zeichen von 128 bis 255 enthalten jeweils die gleichen Zeichen in inverser Darstellung.

2.4 Betriebssystem

Das Betriebssystem ("Operating System", OS) des ATARI umfasst unter anderem die Routinen für den Kalt- und Warmstart, für Ein- und Ausgabe,

Fließkommaroutinen und Routinen zur Abarbeitung verschiedener Interrupts. Beim ATARI 400/800 liegt das Betriebssystem im Speicherbereich zwischen 55296 ($D800) und 65535 ($FFFF) und ist genau 10 KByte lang.

Beim ATARI 600XL, 800XL, 800XE, 65XE, 130XE und XEGS nimmt es zusätzlich den Bereich zwischen 49152 ($C000) und 53247 ($CFFF) ein. Zusätzlich gibt es noch den Selbsttest, der normalerweise unter den Hardware-Registern "versteckt" liegt und nur bei Bedarf im Bereich von 20480 ($5000) bis 24575 ($57FF) eingeblendet wird. Insgesamt ist es also genau 16 KByte lang.

Bis zur Einstellung im Jahr 1992 hatte es von ATARI eine Reihe an offiziellen Betriebssystemversionen gegeben. Man kann die Betriebssysteme der XL/XE-Rechner am Inhalt der Speicherstellen 65526, 65527 ($FFF6, $FFF7) erkennen. Dabei gibt $FFF6 den Funktionsumfang an (0=ohne, 1=mit Parallelbusunterstützung) und $FFF7 die eigentliche Versionsnummer. In der nachfolgenden Tabelle steht unter 'Modell' der jeweils erste ATARI, in dem die Version zum Einsatz kam.

Modell	**OS-Version**	**$FFF6**	**$FFF7**	**Kommentar**
400/800	A, B	255	255	4 Varianten: A, B in NTSC, PAL
1200XL	10	0	10	ohne Parallelbus-unterstützung
1200XL	11	0	11	Korrekturen der Version 10
600XL	1	0	1	nur in den allerersten 600XL
600XL/800XL	2	1	2	mit Parallelbusunterstützung
130XE	3	1	3	mit Korrekturen beim Selbsttest
XEGS	4	1	4	mit Missile-Command-Unterstützung

Als Nachfolger für den ATARI 400 und 800 erschien zunächst der ATARI 1200XL, der in den meisten Belangen dem 800XL sehr ähnlich war. Es gibt aber einige wichtige Unterschiede. Der 1200XL hatte kein eingebautes BASIC, keinen parallelen Bus, dafür aber vier zusätzliche Funktionstasten und zwei steuerbare Status-LEDs. Statt des "MEMO-PAD" und



Abb. 2.10: ATARI 400



Abb. 2.11: ATARI 800



Abb. 2.12: ATARI 1200XL

zusätzlich zum eingebauten Selbsttest, gibt es beim 1200XL einen Titelbildschirm mit einem wunderschönen ATARI-Markenzeichen. Auch für dieses Modell existieren zwei verschiedene Betriebssystemversionen, nämlich Revision "A" und "B". Da dieser Rechner nur als NTSC-Version gebaut wurde und in Europa gar nicht erst auf den Markt kam, existiert er hier nur in wenigen Einzelexemplaren. Selbst in USA und Kanada wurde er nicht in großen Stückzahlen verkauft. Daher soll er hier nicht weiter beachtet werden. Mit einiger Vorsicht lässt sich aber sagen, dass die hier vorliegenden Informationen bedingt auch auf den 1200XL zutreffen.

Kurz darauf erschienen in 1983 der ATARI 600XL und 800XL. Das größte Manko des 1200XL, die fehlende Ausbaumöglichkeit, war durch den zugänglichen Systembus beseitigt. Weitere zusätzliche Merkmale sind die Selbsttestfunktion und das eingebaute BASIC.

Weggefallen dagegen sind leider die Funktionstasten (ohne ersichtlichen Grund) und die Status-LEDs, deren Steuerleitungen nun für die Speicherverwaltung benötigt werden.

Im Übrigen soll das Betriebssystem des 1200XL weitgehend mit dem 600XL und 800XL übereinstimmen. Dass im Selbsttest noch immer die vier zusätzlichen Funktionstasten des 1200XL berücksichtigt werden, liegt aber wohl daran, dass schließlich auch noch ein ATARI 1400XL und ein 1450XLD erscheinen sollten, die ebenfalls diese Funktionstasten vorzuweisen hatten. Ein anderes Beispiel: Der Tastaturinterrupt fragt auf

einen bestimmten Tastencode ab, mit dem der Tastenklick abgeschaltet werden konnte. Die dazu benötigte Taste ist leider verschwunden (kann aber simuliert werden → Keyboard).



Abb. 2.13: ATARI 600XL



Abb. 2.14: ATARI 800XL

1985 kam mit dem 65XE (bei uns als 800XE) der Nachfolger des 800XL auf den Markt. Beide Geräte konnte man eine Weile gleichzeitig kaufen.



Abb. 2.15: ATARI 65XE (baugleich mit 800XE)

Der 800XE gleicht (intern) fast vollständig dem 800XL. Bisher wurde noch kein Unterschied zum Betriebssystem des 800XL festgestellt; von ATARI-Seite wurde auch bestätigt, dass das Betriebssystem nicht verändert worden ist. Im Zuge der Sparmaßnahmen ist der Systembus auf das ECI (Enhanced Cartridge Interface) gestutzt worden. Alle weggefallenen Leitungen liegen aber am Modulschacht an.

Gleichzeitig kam als großer Bruder des 800XE der 130XE auf den Markt, dessen einziger Unterschied im verdoppelten RAM-Speicher (128 KByte) liegt. Rein äußerlich kann man die verschiedenen XE-Modelle 65, 800 und 130 nur an der Beschriftung unterscheiden.

Für den 130XE gibt es ein angepasstes OS, das im Selbsttest 128 KByte statt der üblichen 64 KByte prüft und anzeigt. Eine offizielle Dokumentation von ATARI ist dazu nicht bekannt.

Eine Besonderheit stellt in diesem Zusammenhang der mit arabischem Betriebssystem ausgestattet 65XE dar, der aber ebenso wie die 1400er XL-Modelle nie vermarktet wurde.

Als ATARIs letztes Modell mit der 8-Bit-CPU 6502 erschien im Jahre 1987 das "XE Game System", kurz XEGS genannt.



Abb. 2.16: XE Game System (Komplettpaket)

Das XEGS wurde in verschiedenen Konfigurationen verkauft und enthielt im Komplettpaket eine abnehmbare Tastatur, eine Light Gun sowie einen dem Design angepassten Joystick CX-40. Als Besonderheit ist das Spiel 'Missile Command' im ROM eingebaut, wodurch das OS-ROM nun auf 32 KByte Größe verdoppelt ist. Gleichzeitig wurde damit die Nutzung der MMU verändert, sodass Änderungen an der Hardware nicht so wie bei den XL/XE-Modellen ausgeführt werden können. Das eingebaute Missile Command startet bei abgenommener Tastatur automatisch mit dem Einschalten, wenn keine andere Cartridge eingesteckt ist.

Durch die Veränderungen des Betriebssystems gibt es naturgemäß Probleme mit der Kompatibilität (siehe dazu auch unter "Kompatibilität").

Im Zuge der XE-Modellreihe wurde das OS noch einmal leicht verändert. Die Änderungen machen sich im Normalbetrieb jedoch nicht bemerkbar.

Die Unterschiede im internen Aufbau der verschiedenen Modelle werden in Kapitel 4 gezeigt.

Das kommentierte Listing des Betriebssystems des ATARI 400 und 800 war direkt von ATARI erhältlich (siehe Quellennachweis), während meines Wissens ein offizielles Listing des XL- und XE-Betriebssystems nicht erhältlich ist. Es existieren allerdings einige disassemblierte und kommentierte Listings des Betriebssystems vom ATARI 800XL.

Die Fließkommaroutinen liegen zwar im Speicherbereich des Betriebssystems, gehören aber vom Ursprung her zum BASIC und sind daher zusammen mit ATARI-BASIC als dokumentiertes Listing im Buch "Inside ATARI-BASIC" abgedruckt (siehe Quellennachweis).

Unterschiede gibt es übrigens auch beim ATARI-BASIC, von dem die Versionen A, B und C existieren. Version A befindet sich in den BASIC-Cartridges, die man für die 400/800-Modelle und den 1200XL benötigt. Die Version B wurde 1983 in den XL-Modellen und die Version C ab 1984 in den XL/XE-Modellen sowie dem XEGS verbaut. Die Revision C gilt allgemein als am wenigsten mit Fehlern behaftet. Die verschiedenen Versionen lassen sich durch Auslesen der Speicherstelle 43234 ($A8E2) wie folgt identifizieren:

162 → Revision A
96 → Revision B
234 → Revision C

Also mithilfe des einfachen BASIC-Befehls

```
PRINT PEEK(43234)
```

kann man die im eigenen ATARI befindliche BASIC-Revision ermitteln und bei Bedarf durch Austausch des BASIC-ROMs aktualisieren.

2.5 Bildschirmtreiber

Zur Ansteuerung der Grafik verfügt das Betriebssystem des ATARI über einen integrierten Gerätetreiber (S:). Er unterstützt eine Vielzahl verschiedener Grafikmodi im Schreibe- und Lesebetrieb.

Folgende CIO-Funktionen werden vom Bildschirmtreiber unterstützt:

OPEN
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS
PUT RECORD
GET STATUS (Dummy-Funktion)

Zusätzlich werden folgende Kommandos unterstützt:

DRAW
FILL

Der Bildschirmtreiber kann 28 verschiedene Grafikmodi erzeugen, und zwar die Grafikstufen 1 bis 8 und 12 bis 15 sowohl mit als auch ohne Textfenster und die Modi 0 und 9 bis 11, bei denen ein Textfenster nicht unterstützt wird. Die Übersichtstabelle und nachfolgend die Beschreibung der einzelnen Grafikstufen.

Anzeigeart	Nummer	Farben	Größe	Platzbedarf
normaler Text	0	1 1/2	40*24	992 Bytes
Großtext	17	5	20*24	672 Bytes
	1	5	20*20	674 Bytes
	18	5	20*12	420 Bytes
	2	5	20*10	424 Bytes
Vierfarbgrafik	19	4	40*24	432 Bytes
	3	4	40*20	434 Bytes
	21	4	80*48	1176 Bytes
	5	4	80*40	1174 Bytes
	23	4	160*96	4200 Bytes
	7	4	160*80	4190 Bytes
	31	4	160*192	8138 Bytes
	15	4	160*160	8112 Bytes
Zweifarbgrafik	20	2	80*48	696 Bytes
	4	2	80*40	694 Bytes
	22	2	160*96	2184 Bytes
	6	2	160*80	2174 Bytes
	30	2	160*192	
	14	2	160*160	
Hochauflösende Grafik	24	1 1/2	320*192	8138 Bytes
	8	1 1/2	320*160	8112 Bytes
GTIA-Modi	9	16	80*192	8138 Bytes
	10	9	80*192	8138 Bytes
	11	16	80*192	8138 Bytes
Vierfarbtext	28	4	40*24	992 Bytes
	12	4	40*20	990 Bytes
	29	4	40*12	
	13	4	40*10	

Tab. 2.5: Übersicht der Grafikstufen

Wichtig:
- Grafikstufen 1-8 und 12-15 enthalten im Normalmodus zu sätzlich unten ein vierzeiliges Textfenster.
- Grafikstufen 12-15 sind nur auf XL/XE-Geräten ansprechbar.

2.5.1 GRAPHICS 0

Dies ist der normale Textmodus, in den man beim Einschalten gelangt. Er wird vom Editor (Gerätetreiber E:) normalerweise beim Programmieren wie folgt benutzt: Einrichtung von 24 Zeilen zu je 40 Zeichen. Diese theoretische Bildschirmbreite wird allerdings vom Betriebssystem auf 38 Zeichen beschränkt, um sicherzustellen, dass auch alle Zeichen auf dem Bildschirm zu erkennen sind. Die Begrenzung der nutzbaren Spalten kann man über die Systemvariablen LMARGN (linker Rand, 82; $52) und RMARGN (rechter Rand, 83; $53) verändern (→ Speicherplan).

Die Stelle, an der das nächste Zeichen ausgegeben wird, wird durch einen Cursor markiert. Der Cursor ist jeweils die inverse Darstellung des Zeichens, über dem er steht (zu Beginn also ein inverses Leerzeichen). Der Cursor kann auch unsichtbar gemacht werden. Dazu muss man CRSINH (752; $2F0) auf einen Wert ungleich 0 setzen.

Intern ist der Textbildschirm nicht aus physikalischen Zeilen (das sind die Zeilen, wie man sie auf dem Bildschirm sieht), sondern aus logischen Zeilen aufgebaut. Logische Zeilen können bis zu 120 Zeichen oder bis zu dreimal so lang wie eine physikalische Zeile werden. Die ist auch der Grund dafür, das beispielsweise unter BASIC eine Programmzeile nur bis zu 120 Zeichen fassen kann. Zur Kontrolle, an welcher Stelle des Bildschirms eine logische Zeile beginnt, benutzt das Betriebssystem die Tabelle LOGMAP (690; $2B2). Ein zusätzliches Textfenster wird in diesem Modus nicht vom Betriebssystem unterstützt. Durch Setzen von BOTSCR (703; $2BF) auf 4 kann man allerdings erreichen, dass nur noch in den untersten vier Zeilen ediert werden kann. Zusätzlich muss man den Editor (E:) und den Bildschirmtreiber öffnen, sodass man über den Bildschirmtreiber auf die oberen 20 Zeilen und über den Editor auf das Textfenster zugreifen kann.

Die XL/XE-Geräte unterstützen zusätzlich im Textmodus das sogenannte Fine-Scrolling, bei dem die Zeilen nicht zeichenweise, sondern punktweise nach oben bewegt werden. Diesen Modus erreicht man, indem man vor Ausführung des OPEN-Kommandos (in BASIC: GRAPHICS) das Flag FINE (622; $26E) auf den Wert 255 setzt.

2.5.2 GRAPHICS 1 und 2

Diese beiden Textmodi bieten 20*24 bzw. 20*12 Zeichen (ohne Textfenster) in vier verschiedenen Textfarben und einer Hintergrundfarbe. Obwohl hier auf dem Bildschirm Text dargestellt wird, behandelt der Bild-

schirmtreiber diese Modi wie Grafik und unterstützt daher auch keine logischen Zeilen, Scrolling etc.

Zeichen werden in einem verkürzten ATASCII-Code übergeben, bei dem die obersten zwei Bits die Nummer des zuständigen Farbregisters und die restlichen sechs Bits das Aussehen des Zeichens bestimmen (Tab. 2.6).

2.5.3 GRAPHICS 12 und 13

Bei diesen beiden Textgrafikstufen handelt es sich um Vierfarb-Textgrafik. Während bei Modus 12 die Zeichen die gleiche Größe haben wie im normalen Textmodus, haben sie im Modus 13 die doppelte Höhe. Diese Textmodi unterscheiden sich sowohl im Datenformat als auch in der Darstellung der Zeichen von den Modi 1 und 2, bei denen jeweils die Farbe für das gesamte Zeichenfeld durch die obersten Bits im Code festgelegt wird. Bei den Vierfarbmodi werden die Zeichen genau wie in GRAPHICS 0 übergeben und abgespeichert. Der Unterschied liegt darin, wie die Zeichen auf dem Bildschirm aussehen. Während im normalen Textmodus jedes gesetzte Bit im Zeichensatz auch genau einen Punkt auf dem Bildschirm anschaltet, werden in diesem Modus für jeden Punkt zwei Bits benötigt. Da zwei Bits vier verschiedene Zahlenwerte annehmen können, kann jeder Punkt in vier verschiedenen Farben gesetzt werden. Damit verringert sich die Anzahl der Punkte in der Horizontalen natürlich von acht auf vier.

Ein Sonderfall sind die "inversen" Zeichen, also die Zeichen, bei denen das höchste Bit gesetzt ist. Bei diesen Zeichen nehmen die Punkte, für die beide Bits gesetzt worden sind, einen fünften Farbton an (Tab. 2.11).

2.5.4 GRAPHICS 3-11, 14, 15

Dies sind die normalen Grafikmodi, die sich nur jeweils durch die Auflösung und die Anzahl der Farben unterscheiden. Eine Sonderstellung nehmen allerdings die Modi 9, 10 und 11 ein, bei denen die Darstellung eines Textfensters nicht vom Betriebssystem unterstützt wird.

2.5.5 Funktionen des Bildschirmtreibers

Die Beschreibung der einzelnen Funktionen des Bildschirmtreibers:

OPEN

Der Gerätename ist für den Bildschirmtreiber "S:"; Gerätenummern und Dateinamen werden ignoriert.

Über AUX1 (→ CIO) können folgende Optionen eingestellt werden:

Bit 5 (gesetzt): Der Bildschirmspeicher wird nicht gelöscht. Auf diese Art und Weise hat man die Möglichkeit, den gleichen Bildschirminhalt in verschiedenen Grafikstufen anzusehen.

Bit 4 (gesetzt): Ein Bildschirmfenster im normalen Textmodus wird in den vier untersten Zeilen eingeblendet (nicht in GRAPHICS 0 und 9 – 11).

Bit 3 (gesetzt): Kanal für Schreiben öffnen.

Bit 2 (gesetzt): Kanal für Lesen öffnen.

Alle diese Optionen können untereinander kombiniert werden.

Über AUX2 wird die Nummer der Grafikstufe (0 bis 15) ausgewählt. Die Angabe der Grafikstufe unterscheidet sich insofern vom GRAPHICS-Befehl in BASIC, als die Information über die eigentliche Grafikstufe und die Optionen (Bildschirmfenster und Nicht-Löschen des Bildspeichers) getrennt übergeben werden.

Das Betriebssystem legt den für das Bild benötigten Speicher direkt unter RAMTOP (106; $6A). Da außerdem je nach Grafikstufe die Anzahl der benötigten Bytes schwankt, überprüft das Betriebssystem zunächst, ob genug Speicherplatz vorhanden ist.
Das aufrufende Programm sollte vor dem OPEN-Kommando APPMHI (14; $E) auf das höchste belegte Byte setzen. Wenn das Öffnen des Bildschirms mehr Speicherplatz benötigen sollte, als zwischen APPMHI und RAMTOP zur Verfügung steht, wird der Fehlercode 147 ($93, zu wenig Speicherplatz) erzeugt.

Nach Rückkehr von der OPEN-Operation enthält MEMTOP (741; $2E5) die Adresse des letzten freien Bytes im Speicher. Außerdem werden die Register CRSINH (Sichtbarkeit des Cursors), die Farbregister und die Tabelle mit den Positionen für Tabulatorstopps (→ Editor) initialisiert.

CLOSE

Es wird lediglich der benutzte IOCB (→ CIO) für eine anderweitige Benutzung freigesetzt.

GET CHARACTERS und GET RECORD

Mittels dieser Funktionen kann man Daten vom Bildschirm lesen. GET RECORD funktioniert jedoch nur mit dem Editor - beim Bildschirmtreiber

kommt immer ein Error 141, da dieser kein EOL-Zeichen liefert. Dabei wird jeweils von der laufenden Cursorposition das entsprechende Zeichen bzw. die Farbnummer eingelesen und der Cursor um ein Zeichen (bzw. einen Punkt) weiterbewegt. Die Daten sind dabei jeweils genau wie bei PUT CHARACTERS abgespeichert.

PUT CHARACTERS und PUT RECORD

Über diese Funktionen können Daten auf dem Bildschirm ausgegeben werden. Die Codierung der Daten unterscheidet sich für die einzelnen Grafikstufen erheblich (siehe Tabellen).

Das Setzen eines einzelnen Punktes ähnlich dem BASIC-Befehl PLOT wird durch Setzen der Cursorposition über COLCRS (84; $54) und ROWCRS (85; $55) und Ausgaben eines einzelnen Zeichens vorgenommen.

Datencodierung in GRAPHICS 0

In der normalen Textgrafik werden alle Zeichen im ATASCII-Code überge-ben (→ Gesamttabelle in Kapitel 3).

Datencodierung in GRAPHICS 1 und 2

COLOR in GRAPHICS 1 und 2						
Farbregister				**Zeichensatz**		
0	**1**	**2**	**3**	**224**	**226**	**206**
32	0	160	128		♥	á
33	1	161	129	!	┣	ù
34	2	162	130	"	▕	Ñ
35	3	163	131	#	┛	É
36	4	164	132	$	┫	ç
37	5	165	133	%	┓	ô
38	6	166	134	&	╱	ò
39	7	167	135	'	╲	ì
40	8	168	136	(	◢	£
41	9	169	137	)	▗	ï
42	10	170	138	*	◣	ü
43	11	171	139	+	▝	ä
44	12	172	140	,	▘	Ö
45	13	173	141	-	▔	ú
46	14	174	142	.	▁	ó
47	15	175	143	/	▖	ö
48	16	176	144	0	♣	Ü
49	17	177	145	1	┏	â
50	18	178	146	2	━	û
51	19	179	147	3	╋	î
52	20	180	148	4	●	é
53	21	181	149	5	▄	è
54	22	182	150	6	▎	ñ
55	23	183	151	7	┳	ê
56	24	184	152	8	┻	ȧ
57	25	185	153	9	▌	à
58	26	186	154	:	┗	Ȧ
59	27	187	-	;	␛	␛
60	28	188	156	<	↑	↑
61	29	189	157	=	↓	↓
62	30	190	158	>	←	←
63	31	191	159	?	→	→

Farbregister				**Zeichensatz**		
0	**1**	**2**	**3**	**224**	**226**	**206**
64	96	192	224	@	♦	i
65	97	193	225	A	a	
66	98	194	226	B	b	
67	99	195	227	C	c	
68	100	196	228	D	d	
69	101	197	229	E	e	
70	102	198	230	F	f	
71	103	199	231	G	g	
72	104	200	232	H	h	
73	105	201	233	I	i	
74	106	202	234	J	j	
75	107	203	235	K	k	
76	108	204	236	L	l	
77	109	205	237	M	m	
78	110	206	238	N	n	
79	111	207	239	O	o	
80	112	208	240	P	p	
81	113	209	241	Q	q	
82	114	210	242	R	r	
83	115	211	243	S	s	
84	116	212	244	T	t	
85	117	213	245	U	u	
86	118	214	246	V	v	
87	119	215	247	W	w	
88	120	216	248	X	x	
89	121	217	249	Y	y	
90	122	218	250	Z	z	
91	123	219	251	[	♠	Ä
92	124	220	252	\	|	
93	125	221	253	]	↖	
94	126	222	254	^	◀	
95	127	223	255	_	▶	

Tab. 2.6: COLOR in GRAPHICS 1 und 2

Datencodierung in GRAPHICS 3-8, 14+15

COLOR in GRAPHICS 8	
Farbregister	Farbnummer
0	nicht benutzt
1 (Helligkeit)	1
2	0
3	nicht benutzt

Tab. 2.7: COLOR in GRAPHICS 8

COLOR in GRAPHICS 4,6,14	
Farbregister	Farbnummer
0	1
1	unbenutzt
2	unbenutzt
3	unbenutzt
4	0

Tab. 2.8: COLOR in GRAPHICS 4,6 und 14

COLOR in GRAPHICS 3,5,7,15	
Farbregister	Farbnummer
0	1
1	2
2	3
3	unbenutzt
4	0

Tab. 2.9: COLOR in GRAPHICS 3,5,7 und 15

Datencodierung in GRAPHICS 9-11

COLOR in GRAPHICS 10	
Farbregister	Farbnummer
Adresse 704	0
Adresse 705	1
Adresse 706	2
Adresse 707	3
0	4
1	5
2	6
3	7
4	8

Tab. 2.10: COLOR in GRAPHICS 9-11

Die Grafikstufen 9 und 11 sind insofern Spezialfälle, als die 16 verschiedenen Farben nicht über einzelne Farbregister festgelegt werden. Vielmehr beeinflusst ein Farbregister, Farbregister 4, alle verfügbaren Farben. In GRAPHICS 9 wird mit diesem Register die Farbe festgelegt. Man hat dann 16 verschiedene Helligkeitsstufen (Farben 0 - 15) dieser Farbe zur Auswahl. In GRAPHICS 11 dagegen wird mit Farbregister 4 eine Helligkeit festgelegt. Man kann dann alle 16 Farbtöne (Farben 0-15) in dieser einen Helligkeit benutzen.

Datencodierung in GRAPHICS 12+13

COLOR in GRAPHICS 12 und 13	
Farbregister	Bitkombination des Punktes
0	01
1	10
2	11 (normal)
3	11 (invers)
4	00

Tab. 2.11: COLOR in GRAPHICS 12 und 13

Nach Ausgabe jedes Zeichens (Punktes) wird der Cursor um ein Zeichen nach rechts bewegt und gegebenenfalls eine neue Zeile begonnen. Lediglich im normalen Textmodus werden dabei die Register LMARGN (82; $52) und RMARGN (83; $53) berücksichtigt.

Eine Sonderstellung nehmen zwei Zeichencodes ein:

- Wird der Wert 125 ($7D) ausgegeben, wird unabhängig von der Grafikstufe der Bildschirm gelöscht.

- Der Wert 155 ($9B) wird auch vom Bildschirmtreiber als Zeilenendezeichen interpretiert.

Sämtliche Bildschirmausgaben können mit CONTROL-1 angehalten und wieder gestartet werden.

STATUS

Diese Funktion liefert stets den Wert 1 (Status OK).

DRAW (Kommandonummer: 17, $11)

Mit dieser Funktion ist es möglich, von der letzten Cursorposition eine Linie zu der in COLCRS und ROWCRS angegebenen Position zu ziehen. Falls nicht vorher das Register ATACHR (763; $2FB) auf einen neuen Farbwert gesetzt wird, wird die zuletzt benutzte Farbe weiterverwendet.

FILL (Kommandonummer: 18, $12)

Dieses spezielle Kommando ermöglicht es, eine Fläche mit einer bestimmten Farbe zu füllen. Zunächst muss man dazu die rechte Begrenzung der auszufüllenden Fläche mit DRAW zeichnen. Anstatt für die linke Begrenzung den DRAW-Befehl zu verwenden, benutzt man nun jedoch das FILL-Kommando. Dabei wird so lange links nach rechts gezeichnet, bis man auf einen Punkt trifft, der nicht die Hintergrundfarbe hat.

Folgende Fehler können bei der Verwendung des Bildschirmtreibers auftreten:

141, $8D — Cursor außerhalb des Bildschirms.

145, $91 — nicht existierende Grafikstufe (kann nur bei ATARI 400/800 auftreten, da bei diesen Geräten das Betriebssystem die Grafikmodi 12-15 nicht ansteuern kann).

147, $93 — nicht genug Speicherplatz für die ausgewählte Grafikstufe vorhanden.

Nummer	Farbwert		Bezeichnung
Dez	Hex	Dez	
0	00	0	Grau
1	10	16	Gold
2	20	32	Orange
3	30	48	Rot
4	40	64	Rosa
5	50	80	Violett
6	60	96	Purpur
7	70	112	Blau
8	80	128	Blau
9	90	144	Hellblau
10	A0	160	Türkis
11	B0	176	Blaugrün
12	C0	192	Grün
13	D0	208	Gelbgrün
14	E0	224	Orangegrün
15	F0	240	Hellorange

Tab. 2.12: Farbtabelle

2.6 ATARI XEP80

Die 1987 erschienene XEP80 ist eine externe, monochrome 80-Zeichen-Grafikkarte. Sie wird an Controller Port 1 oder 2 angeschlossen und verfügt über eine eigene Stromversorgung. Alle Funktionen werden über einen zu ladenden Handler (→ Gerätetreiber) gesteuert. Die serielle Datenübertragungsrate über den Controllerport beträgt 15625 Bits pro Sekunde. Zusätzlich verfügt die XEP80 über ein Centronics-Interface mit 2 KByte Puffer zum Anschluss von Druckern.



Abb. 2.17: ATARI XEP80

Im Textmodus werden 25 Zeilen zu je 80 Zeichen angezeigt, wobei die tatsächliche Zeilenlänge 256 Zeichen beträgt und in Abhängigkeit von der verwendeten Software horizontal gescrollt werden kann. Ein Zeichen ist 7 Pixel breit und 10 Pixel hoch.

Im Grafikmodus werden 320x200 Pixel dargestellt. Die Auflösung entspricht dem Grafikmodus 15 des ANTIC (→ Punktgrafikstufen), bietet aber acht Zeilen mehr. Insofern werden also die zur Verfügung stehenden Grafik- und Text-Modi um je einen ergänzt.

Der von ATARI mitgelieferte Handler ersetzt beim Booten die Vektoren für E:, S:, und P: in der Handleradressentabelle (794;$31A). Per Software kann der ATARI sowohl den ANTIC/GTIA-Bildschirm als auch die Ausgabe zur XEP80 quasi parallel steuern. Ein interessantes Beispiel dafür ist das Terminalprogramm "BobTerm" von Robert Puff, das beide Bildschirme während des Terminalbetriebes nutzt.

ATARI liefert auf der beiliegenden Diskette nicht nur den Handler, sondern auch den Sourcecode zu Handler und Relocator mit.

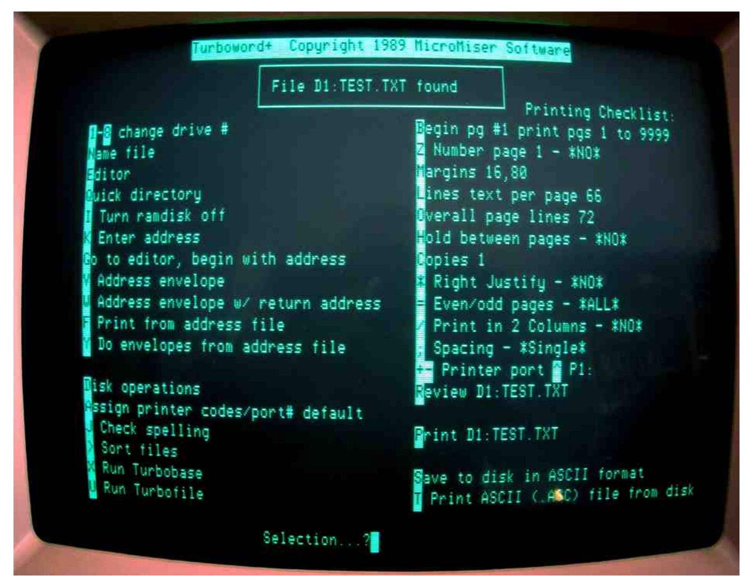


Abb. 2.18: Textmodus: Turboword+ von MicroMiser

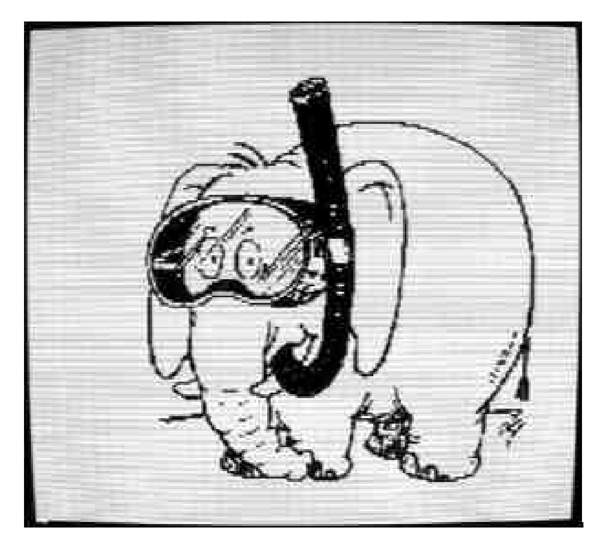


Abb. 2.19: Grafikmodus: 320x200 Pixel

2.7 RESET-Routine und BOOTVORGANG

Ein Bootvorgang (Kaltstart) wird entweder durch Einschalten des Gerätes, Einstecken eines Moduls oder auch unter Umständen durch Drücken der SYSTEM-Reset-Taste ausgelöst. Der Betriebssystemeinsprung für einen Kaltstart liegt bei COLDSV (58487, $E477).

Da der Bootvorgang eine komplizierte Angelegenheit ist, wird hier jeweils der Name der Routine jedem Absatz vorangestellt; Vorgänge, die nur auf XL/XE-Geräten ablaufen, sind durch ein (X) gekennzeichnet.

Der Bootroutine direkt vorgelagert ist die Routine, die das Drücken der RESET-Taste behandelt.

2.7.1 RESET

(X) Zunächst wird überprüft, ob ein Modul eingesteckt oder wieder herausgezogen worden ist, oder ob ein anderes Modul als bei der letzten Überprüfung eingelegt ist. In einem solchen Fall wird zu PWRUP0 verzweigt. Daraufhin wird anhand von COLDST (580; $244) überprüft, ob die Reset-Taste während des Kaltstarts gedrückt worden ist. Wenn ja, verzweigt das Programm nach PWRUP0, ansonsten wird WARMST (8; $8) auf 255 gesetzt und nach PWRUP1 gesprungen.

2.7.2 PWRUP

Hier beginnt die eigentliche Kaltstartroutine.

(X) Der Bootvorgang beginnt zunächst mit einer Warteschleife von rund einer Sekunde. Diese Warteschleife ist eingebaut worden, weil die RESET-Taste bei XL/XE-Geräten ein wenig anders funktioniert. Daraufhin werden (X) die Register PUPBT1 (829; $33D), PUPBT2 (830; $33E) und PUPBT3 (831; $33F) mit den Konstanten 92 ($5C), 147 ($93) und 37 ($25) verglichen. Wenn alle diese Register mit den betreffenden Konstanten übereinstimmen, dann wird nach RESET verzweigt, um festzustellen, ob es sich nicht doch vielleicht um einen Warmstart handelt.

2.7.3 PWRUP0

Hier wird das Flag WARMST (8; $8) auf "0" gesetzt, was nichts anderes als "Kaltstart" bedeutet.

2.7.4 PWRUP1

Zuerst wird das Dezimal-Flag gelöscht und der Stapelzeiger (Stack-Pointer) initialisiert. Daraufhin wird festgestellt, ob ein Spezial-Modul eingelegt ist. Diese Art von Modulen war vermutlich nur für interne Zwecke gedacht, wird aber auch bei einigen Spielen auf Modul verwendet. Spezialmodule sind dadurch gekennzeichnet, dass CARTFG (49149; $BFFD; liegt am Ende des Adressbereiches eines Modules) einen Wert enthält, der über 127 liegt. Ist ein solches Modul eingelegt, dann wird es bereits an dieser Stelle über den Vektor CARTAD (49150; $BFFE) aufgerufen.

(X) Falls kein Spezialmodul eingelegt war, wird an dieser Stelle die Initialisierung der Hardware-Register vorgenommen. (X) Außerdem wird die OPTION-Taste abgefragt und, falls sie gedrückt ist, das BASIC ausgeschaltet, ansonsten angeschaltet. Im Falle eines Warmstartes wird zuvor über BASICF (1016; $3F8) überprüft, ob BASIC angeschaltet gewesen war und das BASIC unter Ignorierung der OPTION-Taste wieder eingeschaltet. Anschließend wird die Anzahl von RAM-Seiten (jede ist 256 Bytes lang) bestimmt und in TRAMSZ (6; $6) abgelegt.

Bei den älteren Modellen erfolgt die Initialisierung der Hardwareregister erst an dieser Stelle. (X) NGFLAG (1; $1) wird vorläufig auf 1 gesetzt, was bedeutet, dass bislang kein RAM-Fehler festgestellt worden ist. Nur im Falle eines Kaltstartes wird nun das gesamte RAM gelöscht, wobei bei den XL- und XE-Modellen zusätzlich ein RAM-Test vorgenommen wird. Wird ein Fehler festgestellt, dann wird NGFLAG gelöscht. Daraufhin wird DOSVEC (10; $A) auf die Routine gesetzt, die erscheint, wenn weder ein Modul eingelegt ist, noch von Diskette oder Kassette gebootet werden kann und auch ATARI-BASIC nicht eingeschaltet ist. Dies ist bei den alten Geräten das MEMO-PAD, bei den neueren Geräten der Selbsttest.

(X) Es folgt eine Prüfsummenüberprüfung des Betriebssystem-ROM. Wiederum wird NGFLAG auf 0 gesetzt, wenn ein Fehler festgestellt wurde.

Nun wird COLDST (580; $244) auf 255 gesetzt (Kaltstart ist im Gange).

Im Falle eines Warmstartes werden noch folgende Speicherbereiche gelöscht:

16-127 ($10-$7F)
512-1004 ($200-$3EC) bei XL und XE, sonst bis 1023 ($3FF)

(X) Es wird überprüft, ob das BASIC eingeschaltet ist und BASICF (1016; $3F8) gesetzt ist (0 bedeutet eingeschaltet, 1 bedeutet ausgeschaltet).

(X) Anschließend werden die Statusbytes PUPBT1 (829; $33D) bis PUPBT3 (831; $33F) gesetzt (siehe oben).

Nun werden die Variablen für die Bildschirmränder, LMARN (82; $52) und RMARGN (83; $53), auf ihre Standardwerte, 2 und 39, gesetzt.

(X) Anhand des Registers PAL (53268; $D014) wird festgestellt, ob es sich um ein PAL-Gerät oder um ein NTSC-Gerät handelt. Abhängig davon (wegen der unterschiedlichen Geschwindigkeit) werden folgende Register initialisiert:

Register	PAL	NTSC
KEYREP (730; $2DA)	5 ($05)	6 ($06)
KRPDEL (729; $2D9)	40 ($28)	48 ($30)
PALNTS (98; $62)	1 ($01)	0 ($00)

Es werden nun die ersten 38 ($26) Bytes der Seite 2 des Speichers, die die Systemvektoren enthält, und die Treibertabelle HATABS (794; $31A) initialisiert.

An dieser Stelle werden weitere Betriebssystemvariablen und sämtliche Peripheriegeräte initialisiert. (X) Außerdem wird NGFLAG überprüft - wenn irgendwann ein RAM- oder ROM-Fehler festgestellt worden war, dann wird nun der Selbsttest aufgerufen.

(X) Danach wird nach PBI-Geräten gesucht: SHPDVS (584; $248) und PDVS (53759; $D1FF) werden der Reihe nach auf die Werte 1, 2, 4, 8, 16, 32, 64, 128 gesetzt, um das ROM vorhandener PBI-Geräte in 55296-57343 ($D800-$DFFF) einzublenden. Enthält 55299 ($D803) den Wert 128 ($80) und 55307 ($D80B) den Wert 145 ($91), wird die Initialisierungsroutine des PBI-ROMs 55321 ($D819) per JSR aufgerufen. Am Ende enthält SHPDVS den Wert 0. Durch Schreiben des Wertes 0 in PDVS wird das PBI-ROM wieder ausgeblendet.

Bei den alten Geräten wird nun zunächst festgestellt, ob im rechten Modulschacht (nur beim ATARI 800!) ein Modul eingelegt ist und dieses gegebenenfalls initialisiert. Dazu wird durch CARTAD (49150; $BFFE bzw. 40957; $9FFD) gesprungen. Das Gleiche passiert daraufhin mit dem "linken" Modulschacht, der bei allen Geräten vorhanden ist.

Nun wird der Bildschirm für den normalen Textmodus geöffnet. Sollte dabei ein Fehler auftreten - die Programmierer des Betriebssystems wissen selbst nicht, wie das passieren könnte - beginnt der gesamte Kaltstartvorgang von Neuem.

Wenn während des Einschaltens die START-Taste gedrückt gehalten worden ist, wird nun vom Kassettenrecorder gebootet und das geladene Programm gestartet. Das ursprüngliche Format von Boot-Dateien beim Kassettenrecorder entspricht dem der Diskettenstation. Beide weisen Datenblöcke bzw. Sektoren von 128 Byte auf; später kamen weitere Formate hinzu, für die diese Aussage nicht mehr gilt.

Es folgt die Überprüfung, ob das oder die (beim 800er) Module das Booten der Diskettenstation und der Geräte im parallelen Bus verlangen. Dazu muss das niederwertigste Bit von CARTFG (49149; $BFFD) gesetzt sein. Wenn keine Module eingelegt sind, entfällt diese Überprüfung, und es wird auf jeden Fall das PBI-Gerät oder die Diskettenstation gebootet.

Nun wird also von Laufwerk 1 gebootet. Dazu wird zunächst der erste Sektor in CASBUF+3 (1024; $400) gelesen. Die ersten sechs Bytes enthalten nähere Informationen über das zu ladende Programm.

Das erste Byte ist dabei unbenutzt, das zweite gibt die Gesamtzahl der zu ladenden Sektoren (bei Kassette Blöcke) an. Das dritte und vierte Byte legt die Adresse fest, an die das Programm geladen werden soll, das fünfte und sechste die Anfangsadresse für die Initialisierung.

Anschließend wird diese Initialisierungsadresse in den Vektor DOSINI (12; $C) übertragen und der erste Sektor an seine Zieladresse verschoben.

Wenn während des Lesens eines Sektors (oder Blocks) ein Fehler aufgetreten ist, dann wird die Meldung BOOT ERROR ausgegeben. Beim Booten von Kassette würde hier der Bootvorgang abgebrochen, beim Booten von Parallelbusgerät oder Diskette wird versucht, den gleichen Sektor noch einmal zu lesen.

Nachdem alle Sektoren (oder Blöcke) eingelesen sind, ruft das Betriebssystem die Unterroutine auf, die genau sechs Bytes hinter der Anfangsadresse beginnt. Diese Routine sollte, falls notwendig, den Bootsvorgang fortsetzen. Natürlich könnte sie auch andere Funktionen erfüllen. Wichtig ist, dass diese Routine dem Betriebssystem mitteilen kann, dass ein Fehler stattgefunden hat, indem sie das Übertrags-Flag des Prozessors setzt. In diesem Fall wird wiederum die Meldung BOOT ERROR auf dem Bildschirm ausgegeben. Wenn das Übertrags-Flag nicht gesetzt ist, wird das geladene Programm über die angegebene Initialisierungsadresse in DOSINI aufgerufen. An dieser Adresse sollte ein Programm stehen, das das geladene Programm zwar initialisiert, aber mit RTS die Kontrolle an das Betriebssystem zurückgibt. Außerdem sollte in diesem Initialisierungsprogramm DOSVEC (10; $A) auf die endgültige Einsprungadresse des Programms gesetzt werden. Handelte es sich um einen Warmstart,

dann wird die gesamte Bootprozedur ausgelassen und lediglich das Programm über DOSINI initialisiert.

Hier wird COLDST (580; $244) wieder auf 0 gesetzt — der Bootprozess ist abgeschlossen.

Schließlich wird überprüft, ob, falls vorhanden, beim eingelegten Modul in CARTFG (49149; $BFFD) Bit 2 gesetzt ist. In diesem Fall wird das Modul durch einen Sprung durch CARTCS (49146; $BFFA) aufgerufen. Beim ATARI 800 findet der gleiche Vorgang zunächst für das linke, dann für das rechte Modul statt.

Wenn kein Modul eingesteckt oder wenn das (die) Modul(e) nicht laufen "wollen", weil das entsprechende Bit in CARTFG nicht gesetzt war, wird das möglicherweise gebootete Programm über DOSVEC (10; $A) aufgerufen.

Und wenn weder ein Modul eingesteckt war (oder das Modul nicht gestartet werden "will"), kein Programm von Kassette gebootet worden ist (oder das initialisierende Programm DOSVEC nicht gesetzt hat) und auch nicht von Diskette gebootet wurde (oder auch hier die Initialisierungsroutine es versäumt hat, DOSVEC zu setzen), dann landet man an dieser Stelle im Selbsttest bzw. im MEMO-PAD, denn auf diese Adresse war DOSVEC zu Beginn gesetzt worden.

(X) Wenn das interne BASIC eingeschaltet ist, wird dieses wie ein Modul angesprungen. Ist das BASIC eingeschaltet und ein Modul aktiv, hat das Modul Vorrang.

2.8 BREAK-TASTE

Die BREAK-Taste erzeugt einen eigenen, aber maskierbaren Interrupt.

Dieser Interrupt setzt das Register BRKKEY (17, $11) auf den Wert 0. Wird die BREAK-Taste während einer I/O-Funktion gedrückt, wird der Fehlercode 128 ($80) erzeugt, sofern die I/O-Funktion keine Wiederholversuche mehr macht. Ist die Taste nicht gedrückt, enthält BRKKEY einen Wert ungleich 0.

Um die BREAK-Taste abzuschalten, muss man sowohl in POKMSK (16; $10) als auch in IRQEN (53774; $D20E) das höchstwertigste Bit löschen.

Vorsicht: *POKMSK und IRQEN werden immer dann zurückgesetzt, wenn ein IOCB für Bildschirmausgabe geöffnet wird, oder wenn nach einem OPEN-Befehl die erste Bildschirmausgabe, z. B. durch PRINT, erfolgt!*

Bei XL/XE-Geräten verfügt die BREAK-Taste außerdem über einen eigenen Vektor, den man auf eine eigene Routine setzen kann (BRKKY, 566,567; $236,$237; nicht zu verwechseln mit dem Vektor für den 6502-BRK-Befehl!).

2.9 CARTRIDGES (ROM-MODULE)

Der ATARI verfügt über einen Anschluss für ROM-Module (beim ATARI 800 zwei). ROM-Module können praktisch beliebig groß sein - allerdings können sie maximal den Speicherbereich zwischen 32768 ($8000) und 49151 ($BFFF), also genau 16 KByte, belegen. Für größere Datenmengen ist das Umschalten zwischen verschiedenen Speicherbänken nötig.

Der zweite, rechte Modulanschluss des ATARI 800 ist praktisch mit dem anderen identisch, mit der Ausnahme, dass hier das Ende des belegbaren Speicherbereichs bei 40959 ($9FFF) liegt.

Jede Cartridge muss über die sechs folgenden Statusbytes am Ende des belegten Adressbereiches verfügen:

Standardmodul		"rechtes" Modul beim ATARI 800
$BFFA $BFFB	Startadresse (low) Startadresse (high)	$9FFA $9FFB
$BFFC	Zwingend 0	$9FFC
$BFFD	Options-Register	$9FFD
$BFFE $BFFF	Initadresse (low) Initadresse (high)	$9FFE $9FFF

Die Startadresse (CARTCS; $BFFA-$BFFB) ist diejenige Adresse, die angesprungen wird, wenn am Ende des Ladevorgangs die Kontrolle endgültig an das Modul übergeben wird.

Das folgende Register (CART) muss stets den Wert 0 haben, damit das Modul korrekt vom Betriebssystem erkannt werden kann.

Über das nächste Register (CARTFG; $BFFD) kann man das genaue Verhalten des Moduls beeinflussen. Insgesamt sind drei Bits wie folgt belegt;

Bit 0 = 0 → Diskette und Geräte am parallelen Bus nicht booten
= 1 → Diskette und Geräte am parallelen Bus booten

Bit 2 = 0 → Modul nur initialisieren
= 1 → Modul initialisieren und starten

Bit 7 = 0 → normales Modul
= 1 → Spezialmodul, wird vor jeder anderen Initialisierung über CARTAD ($BFFE) aufgerufen!

Die letzten beiden Bytes (CARTAD; $BFFE-$BFFF) beinhalten die Adresse der Initialisierungsroutine. Bei Spezialmodulen ist dies die eigentliche Einsprungadresse, bei anderen sollte die Initialisierung mit RTS enden, damit auch das restliche RAM korrekt initialisiert werden kann.

Weitere wichtige Register bei den XL/XE-Geräten für die Arbeit mit Cartridges:

TRIG3 (53267; $D013) dient zur Abfrage des Modulanschlusses. Hat es den Wert 1, dann ist ein Modul eingesteckt und aktiv; 0 bedeutet das Gegenteil.

GINTLK (1018; $3FA) ist das Schattenregister zu TRIG3 und wird innerhalb der internen Vertikal-Blank-Routine mit TRIG3 verglichen. Auf diese Weise soll festgestellt werden, ob ein Modul eingelegt oder herausgenommen worden ist.

Bei jedem Reset ruft das OS des XL/XE über INTINV folgende Routine auf:

```
$C00C  LDA #$40
$C00E  STA $D40E  ;NMIEN
$C011  LDA $D013  ;TRIG3  → Inhalt von Trig3 in das
$C014  STA $03FA  ;GINTLK → Schattenregister kopieren.
$C017  RTS
```

Und anschließend wird bei jedem Vertikal Blank Interrupt geprüft:

```
$C123  LDA $D013  ;TRIG3  → Inhalt von Trig3 mit dem
$C126  CMP $03FA  ;GINTLK → Schattenregister vergleichen.
$C129  BNE $C0DF  ;       → Bei Veränderung springe nach
                            $C0DF in die Endlosschleife im
$C0DF  JMP $C0DF  ;       → OS; nur noch Reset möglich.
```

CARTCK (1003; $3EB) ist die Prüfsumme der Bytes zwischen 49136 ($BFF0) und 49391 ($C0EF).

Die Berechnung der Prüfsumme für den Modulbereich im XL-OS:

```
$C4C9    LDA #$00
$C4CB    TAX
$C4CC    CLC
$C4CD    ADC $BFF0,X
$C4D0    INX
$C4D1    BNE $C4CD
$C4D3    CMP $03EB ;CARTCK
$C4D6    STA $03EB ;CARTCK
$C4D9    RTS
```

CARCTL (54528-54783; $D500-$D5FF), auch CCNTL oder CCTL genannt, ist eine Steuerleitung, die direkt an den Modulanschluss führt und zur Steuerung bestimmter Modulfunktionen dienen kann. Diese Steuermöglichkeit ist auch auf dem ATARI 400 und 800 vorhanden.

Sie wird meist benutzt von Modulen des Typs XEGS (mehr als 16KByte ROM), OSS, MegaCart, Atarimax, Corina, GR8 und noch vielen weiteren. Teilweise besitzen diese Module auch die Möglichkeit, das Modul über eine Adressanwahl oder das Setzen eines Bits im Datenbereich von $D500 bis $D5FF ein- oder auszuschalten. Dabei werden die Signale RD4 und RD5 von +5V auf 0V gesetzt. Das XL/XE-OS registriert dies aufgrund des geänderten Inhalts von TRIG3. Um nicht in der Endlosschleife im OS zu landen, sollte zuvor der Vertikal Blank Interrupt (VBI) gesperrt worden sein. Das Modul "Bounty Bob Strikes Back!" hat die ungewöhnlichste Seitenumschaltung. Das ROM ist 40 KByte groß. Durch Lesen einer Adresse zwischen $8FF7 und $8FF9 bzw. $9FF7 und $9FF9 wird die jeweilige Seite im Bereich $8000-$8FFF oder $9000-$9FFF eingeblendet. Diese Hardwareumschaltung verhinderte lange Zeit eine Analyse des Modultyps.

Als Beispiel zur Erläuterung der Möglichkeiten, die sich durch CARCTL ergeben, sei hier der interne Aufbau von MAC/65 beschrieben.

Es handelt sich dabei um ein Modul von insgesamt 16 KByte Länge. Tatsächlich belegt es im ungünstigsten Fall jedoch nur 8 KByte, im günstigsten Fall sogar gar keinen Speicherplatz! Wie kann das sein?

Der erste "Trick" liegt darin, dass sich das Modul über das Einschreiben eines beliebigen Wertes in Register 54783 ($D5FF) ganz ausblenden lässt. Dadurch wird das RAM von 40900 ($A000) bis 49151 ($BFFF) für andere Anwendungen frei. Eine Nutzungsmöglichkeit für diesen Speicherbereich stellt beispielsweise das Diskettenbetriebssystem der gleichen Firma, DOS XL, dar. Dieses kann sich weitgehend in den Speicherbereich unter dem Modul verschieben und dennoch durch eine ausgeklügelte

Umschaltungstechnik mit dem eingelegten Modul kommunizieren. Diese Technik nutzen auch BeweDOS, RealDOS, SpartaDOS und SpartaDOS X.

Daneben ist das Modul in vier Hauptteile aufgeteilt worden, von denen der wichtigste zwischen 45056 ($B000) und 49151 (9$BFFF) liegt. Die anderen drei Blöcke werden je nach Bedarf im Bereich von 40900 ($A000) bis 45055 ($AFFF) eingeblendet.

Hier die Register zur Steuerung der "OSS Super Cartridges" :

$D500: Bank 1 und Hauptbank einschalten.
$D503: Bank 2 und Hauptbank einschalten.
$D504: Bank 3 und Hauptbank einschalten.
$D5FF: Gesamtes Modul abschalten.

Das Schalten der Bänke wird durch Lesen oder Beschreiben der o.a. Register ausgelöst. Wichtig ist nur der Zugriff auf die Adressen, daher sind die Werte bei einem Schreibzugriff egal! Ein Ansprechen einer der Adressen schaltet die entsprechende Bank ein bzw. das Modul ab. Ist das Modul abgeschaltet, aktiviert ein Zugriff auf eine der Adressen $D500-$D504 die entsprechende Bank und schaltet damit das Modul wieder ein.

Allerdings gibt es da ein Problem: Obwohl die von OSS für die Carts festgelegten Steueradressen $D500-$D50F lauten, verwendet die Cart-Logik zum Dekodieren nur 4 Bits. Daher wirkt sich jeder Zugriff auf eine Adresse $D5xy auf die Cartridge aus, da diese das "x" der Adresse ignoriert. Die OSS-Cart reagiert daher auf jeden Zugriff auf Page $D5. Das kann problematisch sein, falls über Page $D5 noch andere Hardware gesteuert wird wie z.B. eine Echtzeituhr (R Time 8, ARC). Mit SpartaDOS X - weder der alten Version von ICD noch der aktuellen von DLT - gibt es keine Probleme, da die unzureichende Adressdekodierung der OSS-Carts durch SDX aufgefangen wird. Technisch lässt sich das Problem durch Ergänzen eines Chips in der Cart lösen. Informationen dazu gibt es vom ABBUC.

2.10 CIO (Central Input/Output Routine)

Über die CIO , der zentralen Ein- und Ausgabe-Routine, kann im Prinzip die gesamte Ein- und Ausgabe des ATARI gesteuert werden.

Im Folgenden einiges über die Hauptprinzipien der CIO:

Der Zugriff auf die CIO ist völlig unabhängig vom angesprochenen Gerät. Das heißt konkret, dass ein Programm zur Ausgabe auf den Drucker ohne Weiteres auch auf den Bildschirm zugreifen kann. Diese "Kompatibilität"

zwischen Ein- und Ausgabegeräten beschränkt sich natürlich auf die elementaren Ein- und Ausgabefunktionen.

Trotz der Ähnlichkeit in den elementaren Ein-/Ausgabefunktionen können auch problemlos gerätespezifische Funktionen angewendet werden.

Die Ein- und Ausgabe bezieht sich auf sequentielle Dateien. Dateien können aus einer beliebigen Zusammenstellung von Bytes bestehen. Natürlich gibt es auch hierbei Sonderformen wie Programmdateien etc. Die einzige Sonderform, die direkt von die CIO unterstützt wird, ist die aus "Records" bestehende Datei, die jeweils aus bis zu 65534 Zeichen mit nachfolgendem End-of-Line-Zeichen (EOL, Zeilenendezeichen, 155; $9B) besteht. Wie groß Records tatsächlich sein können, hängt vom zur Verfügung gestellten Puffer des Programms ab, das die CIO aufruft - meist ist dieser nur 255 Byte groß wie z.B. in BASIC. Manche Peripheriegeräte unterstützen gleichzeitig mehrere Dateien (von den verbreiteten nur die Diskettenstation), bei allen anderen Geräten ist nur eine Datei vorhanden (beispielsweise der Bildschirminhalt).

Alle Ein- und Ausgabevorgänge laufen über sogenannte Kanäle. Die CIO stellt acht verschiedene Kanäle zur Verfügung, die völlig beliebig irgendeiner Datei zugewiesen werden können. Die einzige Einschränkung besteht darin, dass Kanal 0 beim Einschalten und bei einem Warmstart dem Editor (→ Editor) zugewiesen wird.

Um einen Kanal zu benutzen, muss er zunächst auf die betreffende Datei geöffnet werden. Dazu ist die Angabe einer Dateispezifikation nötig. Eine Dateispezifikation (Filespec) besteht stets aus dem Identifikationszeichen für das Gerät, optional einer Gerätenummer, einem Doppelpunkt und optional einem Dateinamen (Filename). Die genaue Struktur der Dateispezifikation hängt vom Treiber (Programmpaket, das das betreffende Gerät steuert, auch "Handler" genannt) ab. Die CIO wird daraufhin überprüfen, ob das Gerät vorhanden ist, und gegebenenfalls das Medium für den Zugriff auf die angegebene Datei vorbereiten.

Einige CIO-Funktionen arbeiten auch mit noch nicht geöffneten Kanälen. Bei einer solchen Funktion wird automatisch der Kanal geöffnet, die Funktion ausgeführt und der Kanal wieder geschlossen. Diesen Vorgang nennt man "Implied Open" (implizites Öffnen).

Nachdem ein Kanal geöffnet ist, kann man über ihn Daten ein- und/oder ausgeben (je nach Art der Öffnung des Kanals).

Der Datenaustausch wird über reservierte Speicherbereiche mit Datenfeldern oder über den Akkumulator des Mikroprozessors vorgenommen.

Nach Abschluss der Ein- und Ausgabevorgänge muss der Kanal wieder geschlossen werden. Auf diese Weise wird auf dem betreffenden Medium die Datei endgültig installiert (bei Diskettenzugriff wird beispielsweise der Eintrag in die Liste der Programme endgültig gemacht) und der Kanal für einen anderen Verwendungszweck freigemacht.

Jedem Kanal ist ein separater Speicherbereich von 16 Bytes Länge zugeordnet, der Kontrollblock für Ein- und Ausgabe (IOCB, Input/Output Control Block). Die einzelnen Register der IOCBs werden benutzt, um alle wichtigen Funktionen für die CIO bereitzustellen. Beim Aufruf der einzigen Routine (CIOV, $E456, 58484), die für die Verwendung der CIO gebraucht wird, muss das X-Register den Offset (Abstand) des gewünschten IOCB vom Anfang des ersten Blocks enthalten. Da jeder IOCB genau 16 Bytes lang ist, enthält das X-Register also jeweils die Nummer des IOCB multipliziert mit 16. Nach Rückkehr von CIOV enthält das Y-Register den Status des IOCB (eine Liste der möglichen CIO-Fehlermeldungen unter → 2.10.3). Der Status befindet sich außerdem im ICSTA-Byte des IOCB, daneben darf auch das Minus-Flag des Prozessors abgefragt werden.

Ein einfaches Beispiel für einen CIO-Aufruf:

```
NUMMER=1
CIOV=$E456
;
  LDX #NUMMER*16
  JSR CIOV
  BMI ERROR
;falls kein Fehlercode
...
ERROR
  LDX #NUMMER*16
  LDA ICSTA,X
  CMP #128
;mit den einzelnen Fehlercodes vergleichen
...
```

Dieses kleine Beispiel zeigt deutlich, wie praktisch es ist, beim Zugriff die indizierte Adressierung zu verwenden. Auf diese Art und Weise braucht man sich nur Konstanten (oder auch Labels) für den ersten IOCB zu definieren - alle anderen ergeben sich durch Addition des X-Registers (in BASIC wäre eine ähnliche Technik durch Addition einer Variablen X denkbar). Ein weiterer Nebeneffekt ist, dass man so das Programm auch ohne Weiteres für alle anderen Kanäle verwenden kann.

Nun zu den einzelnen Kontrollbytes der IOCBs.

ICHID $340 832
Dieses Byte wird automatisch von der CIO gesetzt und beinhaltet den Index des Gerätenamens in der Treiber-Tabelle (dazu später mehr). Nach Schließung eines Kanals durch CLOSE enthält dieses Byte den Wert 255. Nur bei diesem Wert ist der Kanal also frei.

ICDNO $341 833
Gerätenummer für Gerätetypen, von denen mehrere Exemplare vorhanden sein können (beispielsweise Diskettenstation oder RS-232-Schnittstelle). Auch dieses Byte wird nach einem Öffnungs-Befehl (OPEN) automatisch von der CIO gesetzt.

ICCOM $342 834
Mit diesem Byte wird festgelegt, welche Funktion die CIO ausführen soll. Die Funktionen sind jeweils vom angewählten Gerät abhängig, obwohl die elementaren Funktionen stets die gleiche Nummer haben. Von der Art der Funktion ist auch die Bedeutung der folgenden Bytes abhängig.

ICSTA $343 835
Dieses Byte beinhaltet stets den zuletzt gelieferten Statuscode des Kanals. Es entspricht dem Inhalt des Y-Registers bei der Rückkehr von CIOV.

ICBADR $344,$345 836,837
Dies ist ein Zeiger auf einen speziellen Speicherbereich, der entweder (bei einem OPEN-Befehl) die Dateispezifikation enthält, oder (bei manchen anderen Kommandos) die Adresse der Daten angibt, die gelesen oder geschrieben werden sollen.

Für dieses Register sind auch folgende Benennungen bekannt: ICBAL für $344 und ICBAH für $345!

ICPUT $346,$347 838,839
Dieser Vektor zeigt auf die Routine (dabei Adresse um 1 vermindert) zum Schreiben eines Bytes auf das betreffende Gerät, ansonsten auf eine Routine, die eine Fehlermeldung erzeugt. Dieser Vektor wird zwar vom ATARI-BASIC benutzt, sollte aber in eigenen Programmen nicht verwendet werden.

Für dieses Register sind auch folgende Benennungen bekannt: ICPTL für $346 und ICPTH für $347!

ICBLEN $348,$349 840,841
Dieses Register wird nur für Lese- und Schreibfunktionen benötigt. Der Benutzer muss dabei hier die Anzahl der zu übertragenden Bytes eintragen. Nach Rückkehr aus der CIO enthält es die Anzahl der tatsächlich übertragenen Bytes.

Für dieses Register sind auch folgende Benennungen bekannt: ICBLL für $348 und ICBLH für $349!

ICAUX1 $34A 842
Dies ist das erste der Hilfsregister des IOCB. Die Verwendung hängt ganz vom jeweiligen Gerätetreiber ab. Das OPEN-Kommando benutzt dieses Register, um die Art des Zugriffs festzulegen. Ungeachtet aller Unterschiede für verschiedene Geräte haben hier zwei Bits jeweils eine feste Bedeutung: Bit 3 kennzeichnet eine Ausgabeoperation, Bit 2 eine Eingabeoperation.

Für dieses Byte ist auch folgende Benennung bekannt: ICAX1.

ICAUX2-6 $34B-$34F 843-847
Zweites bis sechstes IOCB-Hilfsregister. Die Funktion hängt vom jeweiligen Gerätehandler ab. Hilfsregister 6 ist bislang unbenutzt und kann wohl als Mittel angesehen werden, die Länge des IOCB auf die "runde" Zahl 16 ($10!) zu bringen.

Für diese Bytes sind auch die Benennungen ICAX2-ICAX6 bekannt).

Einige Funktionen sind für alle Gerätetreiber gleich. Einschränkungen ergeben sich allerdings daraus, dass manche Geräte auf eine einzige Datenrichtung beschränkt sind (Drucker können beispielsweise leider keine Daten einlesen).

2.10.1 Funktionen der CIO

Im Folgenden eine Liste der Funktionen der CIO. Als Code wird das für ICCOM bestimmte Byte bezeichnet. Dabei wird vorausgesetzt, dass jeweils das X-Register den passenden Wert enthält:

Kommando: OPEN → Code: 3
Für die Konstante 3 wird der Name COPN (Command OPeN) empfohlen.
Funktion zum Öffnen eines IOCB. ICBADR muss hierbei die Anfangsadresse der Dateispezifikation enthalten. Der Inhalt von ICAUX1 und ICAUX2 ist vom jeweiligen Gerätetreiber abhängig.

Nach dem Aufruf der CIO können die Register ICHID, ICDNO, ICSTA und ICPUT verändert sein. ICSTA wird den Statuscode der Funktion enthalten, der nach Rückkehr von CIOV auch im Y-Register steht; alle anderen Register sollten sowieso nicht vom Benutzer geändert werden.

Kommando: CLOSE → Code: 12
Für die Konstante 12 wird der Name CCLOSE (Command CLOSE) empfohlen.

Diese Routine schließt den angegebenen Kanal. Dabei werden eventuell noch in einem Puffer vorliegende Daten abgesendet, und der IOCB für eine andere Verwendung freigemacht. Geändert werden durch CLOSE die Register ICHID, ICSTA und ICPUT.

Kommando: GET CHARACTERS → Code: 7
Für die Konstante 7 wird der Name CGBINR (Command Get BINary Record) empfohlen.

Folgende Bytes müssen vom Benutzer gesetzt werden:

ICBADR: Zeiger auf die Stelle, an die der Datenblock gelesen werden soll.
ICBLEN: Anzahl der zu lesenden Bytes.

Diese Routine dient zum Einlesen eines Datenblocks mit einer bekannten Länge an eine bekannte Stelle im Speicher. Hat ICBLEN den Wert 0, wird genau ein Byte eingelesen und bei der Rückkehr von CIOV in den Akkumulator übertragen. Nach Ausführung der Funktion enthält ICSTA den Statuswert und ICBLEN die Anzahl der tatsächlich übertragenen Bytes.

Kommando: PUT CHARACTERS → Code: 11
Für die Konstante 11 wird der Name CPBINR (Command Put BINary Record) empfohlen.

Folgende Register müssen vom Benutzer gesetzt werden:

ICBADR: Zeiger auf den Anfang des zu übertagenden Datenblocks.
ICBLEN: Anzahl der zu übertragenden Bytes.

Diese Routine dient zum Schreiben eines Datenblocks mit bekannter Länge und bekannter Position im Speicher. Wenn ICBLEN auf den Wert 0 gesetzt ist, wird stattdessen der Inhalt des Akkumulators übertragen.

Bei der Rückkehr von CIOV wird nur der Inhalt von ICSTA verändert - bei einem Fehler auch ICBLEN (Anzahl der übertragenen Bytes).

Kommando: GET RECORD → Code: 5
Für die Konstante 5 wird der Name CGTXTR (Command Get TeXT Record) empfohlen.

Folgende Register müssen vom Benutzer gesetzt werden:

ICBADR: Adresse, an die der "Record" gelesen werden soll.
ICBLEN: maximale Anzahl von Bytes, die eingelesen werden sollen.

Bei dieser Funktion werden so lange Zeichen eingelesen, bis entweder die angegebene Zahl eingelesen ist, oder ein End-of-Line-Zeichen ($9B, 155) aufgetreten ist, das einen Record (Datensatz, der durch ein EOL-Zeichen begrenzt ist) begrenzt. Im ersteren Fall wird trotzdem bis zum nächsten EOL-Zeichen weitergelesen und ein "Truncated Record"-Fehler gemeldet.

Nach Rückkehr enthält ICSTA den Statuswert der Funktion, ICBLEN die Anzahl der tatsächlich übertragenen Bytes.

Kommando: PUT RECORD → Code: 9
Für die Konstante 9 wird der Name CPTXTR (Command Put TeXT Record) empfohlen.

Folgende Register sind vom Benutzer zu setzen:

ICBADR: Adresse des zu schreibenden Blocks.
ICBLEN: Maximale Anzahl von zu übertragenden Bytes

Diese Routine dient dazu, einen Record mit der in ICBADR angegebenen Anfangsadresse und der in ICBLEN angegebenen Maximallänge zu schreiben. Enthält der Puffer kein EOL-Zeichen, dann wird dieses automatisch durch die CIO an das Ende des Eintrages gehängt. Nach Rückkehr von CIOV sind ICSTA sowie ICBLEN (Anzahl der übertragenen Bytes) geändert.

Kommando: GET STATUS → Code: 13
Für die Konstante 13 wird der Name CSTAT (Command STATus) empfohlen.

Bei einigen Geräten arbeitet STATUS auch oder nur mit noch nicht geöffneten Kanälen - in einem solchen Fall muss ICBADR auf die betreffende Dateispezifikation zeigen (Implied Open). Nach Rückkehr von CIOV ist ICSTA geändert. Bei einigen SIO-Geräten können in den vier Registern ab DVSTAT ($2EA, 746) weitere Statusinformationen vorliegen.

Kommando: SPECIAL → Code: größer 13
Jeder Kommandocode, der größer als 13 ist, wird als gerätespezifisches Spezialkommando angesehen. Von dieser gerätespezifischen Routine hängt es auch ab, welche Bytes vom Benutzer gesetzt werden müssen und welche Wirkung das hat. Auf jeden Fall muss, falls der betreffende IOCB noch nicht geöffnet ist, in ICBADR die Adresse der Dateispezifikation angegeben werden (Implied Open).

Welche Register nach Rückkehr von CIOV verändert sind, hängt ebenfalls vom betreffenden Gerätetreiber ab. Auf jeden Fall enthält ICSTA wiederum den Statuswert der Operation.

Listen der jeweiligen Spezialfunktionen finden sich in den Abschnitten zum jeweiligen Gerätetreiber.

Dieses Kommando entspricht weitgehend dem XIO-Befehl in BASIC.

2.10.2 Dateispezifikationen

Eine Gerätespezifikation besteht aus bis zu fünf Teilen:

1. Geräteidentifikation (z. B.: C, D, E, K, P, R, S)
2. Gerätenummer (1-8)
3. Doppelpunkt (:)
4. Dateiname (z. B.: CIO.TXT)
5. EOL-Zeichen ($9B, 155)

Das Betriebssystem unterstützt folgende Geräteidentifikationen:

C: Kassettenrecorder
E: Editor
K: Tastatur
P: Drucker
S: Bildschirm

Folgende Geräteidentifikationen werden durch Standardprogramme unterstützt:

D: Diskettenstation (PBI- oder SIO-Laufwerke; auch Festplatten)
R: RS-232

Andere bisher bereits benutzte Geräteidentifikationen:

G: Grafikdruck
M: RAM-Floppy

T: Texttreiber für Pixelgrafik
V: Sprachausgabe

Wie eine Dateispezifikation nun genau aussehen muss, ist vom jeweiligen Gerätetreiber abhängig.

2.10.3 Fehlermeldungen der CIO

Einige Fehlermeldungen werden sofort von der CIO erzeugt, ohne vorher erst einen Gerätetreiber aufzurufen:

Error 129 - IOCB already open
Der gewählte Kanal war bereits geöffnet.

Error 130 - Nonexistent Device
Das durch den Gerätenamen spezifizierte Gerät ist nicht in die Gerätetreibertabelle eingetragen.

Error 131 - IOCB Write-Only
Obwohl der Kanal nur für Ausgabe geöffnet war, wurde ein Lesekommando gegeben.

Error 132 - Illegal handler command
Das angegebene Kommando ist nicht erlaubt, da kleiner als 3.

Error 133 - Device or file not open
Der angegebene Kanal war noch nicht geöffnet.

Error 134 - Bad IOCB number
Es wurde eine Kanalnummer (d.h. ein X-Registerinhalt) angegeben, die nicht erlaubt ist. Es gibt nur die Kanäle 0 bis 7, d.h. nur die X-Registerwerte 0,16,32,48,64,80,96,112 sind beim CIO-Aufruf zulässig.

Error 135 - IOCB Read only
Obwohl der Kanal nur für Eingabe geöffnet war, wurde ein Schreibkommando gegeben.

Error 137 - Truncated Record
Beim Kommando GET RECORD war der gelesene Datensatz länger als die erwartete Maximallänge.

Die nächsttiefere Benutzerebene für Ein- und Ausgabe erreicht man, wenn man selbst einen neuen Gerätetreiber schreibt, oder einen bestehenden verändert (→ Gerätetreiber).

2.11 CONSOLE-TASTEN

Unter den CONSOLE-Tasten versteht man im weiteren Sinne alle Funktionstasten des ATARI, als da wären HELP, START, SELECT, OPTION und RESET. Im engeren Sinne sind es allerdings nur START, SELECT und OPTION, wie man gleich sehen wird.

Nähere Informationen zur HELP-Taste befinden sich unter dem Begriff "HELP-Taste", über die System-Reset-Taste kann man in dem Abschnitt "Bootvorgang" nachlesen.

Zur Abfrage der drei eigentlichen CONSOLE-Tasten steht das Register CONSOL ($D01F, 53279) zur Verfügung. Zu jeder Taste gehört ein Bit, das den Status der Taste beinhaltet. Wie man sieht, sind die CONSOLE-Tasten nicht nur völlig unabhängig von der normalen Tastatur, es können sogar alle Kombinationen von CONSOLE-Tasten abgefragt werden.

Verwendet werden nur die drei niederwertigsten Bits; dabei bedeutet ein gesetztes Bit, dass die Taste nicht gedrückt ist. Die Belegung der einzelnen Bits:

Bit 2: OPTION
Bit 1: SELECT
Bit 0: START

Erfahrungsgemäß ist es jedes Mal die gleiche lästige "Denkarbeit", eine korrekte Abfrageroutine zustande zu bekommen, daher hier eine Vorlage:

```
1000 MSTART=1
1010 MSELECT=2
1020 MOPTION=4
1030 CONSOL=$D01F
1040 ;
1050 LDA CONSOL
1060 TAX
1070 AND #MSTART
1080 BEQ STARTGEDRUECKT
1090 TXA
1100 AND #MSELECT
1110 BEQ SELECTGEDRUECKT
1120 TXA
1130 AND #MOPTION
1140 BEQ OPTIONGEDRUECKT
1150 ;gar keine CONSOLE-Taste gedrueckt!
```

2.12 DOS - Disk Operating System

2.12.1 Grundsätzlicher Aufbau des ATARI-DOS

Alle DOS-Versionen für die verschiedenen ATARI-Computer bestehen im groben aus vier Teilen. Diese vier Teile sind von "unten" nach "oben" (in ihrer Sichtbarkeit für den Anwender) nachstehend aufgeführt:

SIO

Die SIO ("Serial Bus Input/Output Routine"; serielle Ein- und Ausgaberoutine) führt die Übertragung von Daten über den seriellen Bus des Computers durch (→ SIO). Bei der Kommunikation mit der Diskettenstation werden die einzelnen Kommandos (Formatieren der Diskette, Lesen und Schreiben von Sektoren und Statusmeldungen) und die entsprechenden Daten über den seriellen Bus gesandt und empfangen. Sie ist also die unterste Ebene des DOS, da die SIO keinerlei logische Bearbeitung der Daten vornimmt, wie es beispielsweise die Zuordnung zu einem "File" (Datei) wäre. Die SIO liegt im ROM des Betriebssystems und wird vom DOS nach dem Setzen des "Device Control Blocks" (Gerätekontrollblock") durchsprungen.

FMS

Das FMS ("File Management System"; Dateienverwaltungssystem) ist für den logischen Aufbau der Diskette und ihrer Dateien verantwortlich. Es organisiert das "Directory" (Disketteninhaltsverzeichnis), ordnet den Einträgen im Directory die Anfangssektoren der Dateien zu und verwaltet die Belegung der Diskette über die VTOC ("Volume Table Of Contents"; Sektorbelegungstabelle). Da alle Dateien sequentiell aufgebaut sind, reicht es, den Anfangssektor einer Datei anzugeben, da zwei Bytes in jedem Sektor auf den nachfolgenden Sektor zeigen (gilt nur für DOS-2-kompatible DOS).

CIO

Die CIO ("Central Input/Output Routine"; zentrale Ein- und Ausgaberoutine) führt sämtliche Ein-/Ausgabeoperationen auf dem ATARI-Computer durch (→ CIO). Sie erlaubt es, Dateien zu öffnen, zu schließen, Zeichen ein- und auszugeben und spezielle Kommandos auszuführen. Diese Ein- und Ausgabe geschieht über verschiedene Gerätetreiber (z. B. C:, E:, D: ...) und Ein-/Ausgabekontrollblöcke (IOCBs). Im Falle des Gerätetreibers D: wird die Diskettenstation über das DOS angesprochen. Sämtliche Kommandos für diesen Gerätetreiber führt das FMS aus.

DUP

Das DUP ("Disk Utility Package", Diskettenhilfspaket) ist der Teil des DOS, der unter DOS 2.0s und DOS 2.5 als getrenntes Programm von der Diskette nachgeladen wird. Dies geschieht z. B. beim BASIC-Befehl DOS. Das DUP, ein menügesteuertes Hilfsprogramm, ist der für den Anwender sichtbare Teil des DOS. Es ist daher bei den verschiedenen DOS-Versionen unterschiedlich ausgeführt. Es ermöglicht für den Benutzer über das FMS die verschiedenen Diskettenfunktionen wie Formatieren, Laden und Speichern von Maschinenprogrammen und Kopieren. Alle von ATARI produzierten DOS-Versionen sind menügesteuert. Die Texte der Menüs und Syntaxhilfen benötigen viel Speicherplatz. Daher wird das DUP meist von Diskette nachgeladen, wenn es aufgerufen wird.

Von anderen Softwarehäusern (so zum Beispiel OSS oder ICD) gibt es jedoch kommandogesteuerte DOS-Versionen, die wie CP/M oder MSDOS zusätzlich zu den DOS-internen Kommandos auch externe Kommandos in Form von einzelnen Hilfsprogrammen verwenden. Sie benötigen für die internen Kommandos zwar kein nachzuladendes DUP, brauchen aber die externen Hilfsprogramme bei Bedarf auf Diskette. Außerdem muss man die internen Kommandos kennen, da über sie keine Syntaxhilfe angezeigt wird. Die externen Kommandos dagegen verfügen in der Regel über eine Syntaxhilfe, die zumeist mit dem Parameter "/?" aufgerufen werden kann.

2.12.2 Unterschiedliche DOS-Versionen

Das während der Jahre 1980 bis zum Erscheinungstermin der ATARI 1050-Diskettenstation (Anfang 1984 in Deutschland) mit allen Diskettenstationen ausgelieferte DOS war das ATARI DOS 2.0s; es wurde damit zum Standard.

Dieses DOS 2.0s löste das nur kurzzeitig verwendete DOS 1.0 ab, das als erste überhaupt existente DOS-Version noch einige Fehler ("Bugs") enthielt. Außerdem verfügte es nicht über den sogenannten "Burst I/O". Bei Burst I/O handelt es sich um eine spezielle Übertragungsart des Diskettenbetriebssystems, bei der die Put- und Get-Byte-Routinen des Gerätetreibers für die Diskettenstation im FMS die CIO umgehen und die zu übertragenen Speicherblöcke in einem Rutsch aus dem Speicher auf die Diskette schreiben bzw. von der Diskette in den Speicher lesen, bevor die CIO wieder die Kontrolle erhält.

Von OSS, der Firma, die das FMS ("File Management System") entwickelt hat, den Grundbaustein des ATARI-DOS also, gibt es das zum ATARI-DOS 2.0s voll kompatible sogenannte DOS XL. Es hat den Vorteil, kommando-



Abb. 2.20: ATARI 1050

und nicht menügesteuert zu sein (ganz ähnlich wie CP/M oder MSDOS). Da alle komplizierteren DOS-Funktionen wie Kopieren oder Diskatteninitialisierung als eigene Programme existieren, entfällt das automatische Nachladen eines DUP.SYS-Programms. Das gesamte Arbeiten mit "MEM.SAV" ist daher ebenso nicht notwendig. Allerdings müssen die Hilfsprogramme bei Bedarf auf Diskette verfügbar sein. Inzwischen gibt es sogar Versionen von DOS XL, die sich bei den XL/XE-Geräten in den freien RAM-Bereichen von 49152 bis 65535 ($C000-$FFFF) unter das OS legen oder auch in das freie RAM unter eine sogenannte Supercartridge (→ Cartridges) wie MAC/65, ACTION!, BASIC XL. Dies ist sogar bei den alten 400/800er Geräten möglich. Dadurch ist natürlich für den Anwender wesentlich mehr Benutzerspeicher frei.

Nachdem die 1050er Diskettenstationen ausgeliefert wurden, gab es zu diesen Geräten für kurze Zeit das ATARI-DOS 3, das mit einer völlig neuen Diskettenorganisation unter anderem den zusätzlichen Platz auf den 1050er Disketten von 130 KByte nutzen konnte. Dieses DOS 3 ist aus verschiedenen Gründen nie richtig von den Benutzern akzeptiert worden. Diese Tatsache schien auch ATARI zu Ohren gekommen zu sein, da auf der Hannovermesse 1985 das ATARI-DOS 2.5 vorgestellt wurde, das sowohl die 1050er Diskettenstation angemessen unterstützt als auch eine RAM-Disk im 130XE ermöglicht. DOS 2.5 ist kompatibel zu DOS 2.0s. DOS

3 war damit "tot". Deshalb wird hier exemplarisch für ATARI-DOS nur der Diskettenaufbau unter DOS 2.0s vorgestellt.

Daneben gibt es zahlreiche sogenannte DOS-2-kompatible DOS, von denen im deutschsprachigen Raum "TurboDOS XL/XE" und "BiboDOS" weit verbreitet sind. Die aktuellste Weiterentwicklung dieses DOS-Typs stellt "XDOS 2.4" aus dem Jahr 2008 dar, das trotz vieler Funktionen nur 5 KByte Speicher inklusive DUP benötigt.

Nachzutragen bleiben noch leistungsfähigere DOS, die seit dem Jahr 1985 herauskamen und problemlos in der Lage sind, größere Diskettenformate als 130 KByte bzw. Festplattenpartitionen bis zu 32 Mbyte zu verwalten. Sie können im allgemeinen DOS 2.x-Disketten lesen.

SpartaDOS

1984 von ICD vorgestellt bietet SD auf dem ATARI ein MSDOS-ähnliches File System (SpartaDOS File System; SDFS). Es arbeitet wie das große Vorbild mit einer Kommandozeile, stellt Verzeichnisse und Unterverzeichnisse bereit und verwaltet Medien mithilfe eines Bit-Map-Systems ähnlich dem FAT-System von MSDOS. So kann SD auch größere Speicherkapazitäten z.B. von Festplatten nutzen. 2005 erweiterte DLT mit SpartaDOS X 4.4x das SDFS auf Version 2.1.

SD verwaltet maximal 65536 Sektoren, was bei einer Sektorengröße von 256 Byte zu maximal 16 MByte bzw. bei 512-Byte-Sektoren (SDX 4.4x) zu maximal 32 MByte Partitionsgröße führt. SD gibt es in verschiedenen Versionen: entweder disketten- oder hardwarebasiert in einer Cartridge bzw. als intern einzubauende Schaltung. Die Einführung dieses SDFS ist der große Verdienst von ICD. Auch andere DOS (Bewe-DOS, RealDOS) nutzten es, bieten die bei DOS XL beschriebenen Möglichkeiten, sind untereinander kompatibel und booten von PBI-Geräten. SDFS arbeitet nicht sequentiell wie ATARI DOS und ist daher schneller und flexibler.

ATARI DOS XE

Als letztes Diskettenlaufwerk für die 8-Bit-Modellreihe brachte ATARI 1987 das doppelseitige XF551 heraus. Damit standen neben den bisherigen Speicherformaten 90 und 130 KByte nun auch 180 und 360 KByte zur Verfügung. Allerdings schaffte es ATARI nicht, zeitgleich ein DOS dafür zu liefern. Das XF551 wurde mit DOS 2.5 ausgeliefert. Erst 1988 erschien DOS XE 1.00. Es unterstützt Speichererweiterungen, nutzt die Fähigkeiten des XF551, bietet Verzeichnisse und Unterverzeichnisse, Batch-Verwaltung und Datumsstempel. Sein Menü erinnert an DOS 3.

Das XF551 funktioniert aufgrund eines fehlerhaften OS manchmal nicht so, wie es sollte. Bei Problemen empfiehlt sich ein Austausch gegen das HyperXF-OS. Infos dazu gibt es beim ABBUC e.V.



Abb. 2.21: ATARI XF551

MyDOS

Das von WORDMARK Systems (Charles Marslett & Robert Puff) 1988 in der Version 4.50 vorgestellte DOS hat ein FMS, das für Disketten in Single, Medium und Double Density zu DOS 2 kompatibel ist. Es arbeitet ebenfalls mit einem DUP.SYS, stellt Verzeichnisse bereit und ist gut dokumentiert. MyDOS bootet außerdem von PBI-Geräten und kann Festplattenpartitionen bis 16 Mbyte verwalten. MyDOS unterstützt keine Floppy-Speeder, da es für die Verwendung von Festplatten am Parallelbus des XL konzipiert wurde. Robert Puff hatte dazu die "Blackbox" entwickelt, die neben der Nutzung von SCSI-Festplatten noch andere Features wie Druckerpuffer, RAM-Erweiterung und einiges mehr bereitstellte.

2.12.3 Aufbau eines Datenfiles (ATARI DOS 2.x)

Damit das DOS ein Datenfile ("Binary File"), das unter der Menüoption "L - Load Binary File" geladen wird, als solches erkennen kann, muss der Aufbau dieses Files wie folgt aussehen: Die ersten sechs Bytes informieren darüber, in welchen Speicherbereich das Programm oder die Daten geladen werden sollen:

255 ($FF)
255 ($FF)
Anfangsadresse der Daten (low byte)
Anfangsadresse der Daten (high byte)
Endadresse der Daten (low byte)
Endadresse der Daten (high byte)
Es folgen n (=Endadresse–Anfangsadresse+1) Datenbytes

Dies ist ein Datenblock, es können jedoch noch weitere folgen. Dabei ist es möglich, dass die beiden Bytes (255; $FF) am Anfang fehlen.

Werden zwei Bytes nach RUNAD (736,737; $2E0,$2E1) oder INITAD (738,739; $2E2,$2E3) geladen, so handelt es sich bei den gelesenen Daten um ein Maschinenprogramm, das durch die in RUNAD angegebene Adresse gestartet und durch die in INITAD angegebene Adresse initialisiert wird. Durch INITAD wird sofort während des Ladens der Daten gesprungen, durch RUNAD erst nach dem Laden der Daten.

Diskettenstruktur des Datenfiles (DOS 2.x)

Das einzelne Datenfile ist sequentiell aufgebaut, d.h., jeder Sektor eines Files zeigt auf den nachfolgenden. Im Directory, dem Disketteninhaltsverzeichnis, ist nur der Anfangssektor des Datenfiles enthalten. Dieser Anfangssektor wird in einen Puffer gelesen. Dabei sind die Bytes 0-124 ($0-$7C) als Datenbytes verwendet. Die letzten drei Bytes enthalten folgende Informationen:

Byte 125 ($7D): Die obersten sechs Bits (Werte zwischen 0 und 63 ($3F)) geben die Nummer des Files an. Diese Nummer entspricht der Stellung im Directory. Da beim Lesen eines Files diese Nummer ständig gleich sein muss, ist damit eine erhöhte Datensicherheit gegeben.

Die unteren zwei Bits geben das "High-Byte" des folgenden Sektors an. Theoretisch ist also eine Sektorzahl von maximal 1023 ($3FF) möglich. Unter DOS 2.5 und 1050er Diskettenstation werden genau diese 1023 Sektoren unterstützt. Sektor 1024 dient der Fortsetzung der VTOC und die restlichen 16 Sektoren (26 Sektoren/Spur * 40 Spuren = 1040 Sektoren/Diskette) sind nicht belegt.

Byte 126 ($7E): Dieses Byte ist das "Low-Byte" des folgenden Sektors. Das Ende eines Datenfiles wird dadurch gekennzeichnet, dass als folgende Sektornummer 0 angegeben wird. Dieser Sektor 0 existiert physikalisch nicht auf der Diskette, kann also als Endmarkierung verwendet wer-

den. Übrigens ist die im Directory eingetragene Länge nicht von Belang. Sie hat im Grunde also nur optische Funktion.

Byte 127 ($7F): Hier wird die Anzahl der Datenbytes angegeben. Theoretisch kann also auch ein Sektor innerhalb eines Files überhaupt keine Daten enthalten. Maximal ist hier der Wert 125 ($7D) möglich, da die Zahl der Bytes nicht von 0, sondern von 1 an gerechnet wird.

Aufbau des Directory (DOS 2.x)

Für das Directory sind acht Sektoren genau in der Mitte der Diskette reserviert (361-368; $169-$170). Die Mitte ist deshalb genommen worden, um insgesamt eine möglichst geringe durchschnittliche Zugriffszeit zu garantieren. Im Disketteninhaltsverzeichnis sind maximal 64 ($40) Einträge enthalten, für jeden Eintrag also sechzehn Bytes, die wie folgt verwendet werden:

Byte 0 ("Flag"): Jedes Bit dieses Bytes hat, falls es gesetzt ist, eine der folgenden Funktionen:

Bit	Dez	Hex	Funktion
7	128	$80	Eintrag ist gelöscht.
6	64	$40	Eintrag ist in Benutzung.
5	32	$20	Eintrag ist gesichert ("locked").
1	2	$2	Immer gesetzt; dient zur Unterscheidung von der vorgehenden DOS 1.0 Version.
0	1	$1	Eintrag für Ausgabe geöffnet.

Das Ende der Directory wird durch eine 0 in diesem Byte gekennzeichnet, damit nicht bei jeder Directoryausgabe alle acht Sektoren gelesen werden müssen. Bei DOS-2.5-Dateien auf Disketten in Medium Density ist Bit 0 statt Bit 6 gesetzt, wenn die Datei Sektoren ab 720 benutzt. Damit sind solche Dateien unter DOS 2.0s unsichtbar, was der Kompatibilität zu DOS 2.0s dient.

Bytes 1,2 ("Count"): Diese beiden Bytes enthalten in der üblichen Reihenfolge (Low/High Byte) die Länge des Files in Sektoren. Wie schon gesagt, spielt dieser Wert nur beim Auflisten der Directory eine Rolle.

Bytes 3,4 ("Start Sector Number"): Die Nummer des ersten Sektors im File wird hier in der üblichen Reihenfolge (Low/High Byte) angegeben.

Bytes 5-12: Diese acht Bytes geben den Filenamen in ATASCII-Werten an.

Bytes 13-15: Diese drei Bytes enthalten den Namen des Extenders.

VTOC - Volume Table Of Contents (DOS 2.x)

Oder auf Deutsch: Datenträgerinhaltsverzeichnis. Damit beim Speichern von Daten auf der Diskette nicht alle schon existierenden Files durchsucht werden müssen, um festzustellen, welche Sektoren sie belegen, existiert die VTOC. Sie gibt für jeden Sektor der Diskette an, ob er belegt ist oder nicht. Die VTOC liegt im Sektor 360 ($168), unter DOS 2.5 wird im 1050er Modus (Medium Density) Sektor 1024 ($400) so mitbenutzt, dass die ersten 100 Bytes der VTOC aus Sektor 360 stammen und die restlichen 38 Bytes aus Sektor 1024 von Byte 84 bis 121. Das dient der Kompatibilität zu DOS 2.0s. Die ersten zehn Bytes der VTOC haben folgende spezielle Bedeutung:

Byte 0: Dieses Byte ist immer 2, um den Unterschied zu DOS 1.0 zu markieren.

Bytes 1,2: Die Gesamtzahl der ursprünglich freien Sektoren (707; $2C3 oder 1010; $3F2) wird hier in der üblichen Reihenfolge (Low/High Byte) angegeben.

Bytes 3,4: Die Gesamtzahl der noch freien Sektoren auf der Diskette wird hier in der üblichen Reihenfolge (Low/High Byte) angegeben. Bei DOS 2.5 steht hier nur die Anzahl der freien Sektoren bis Sektor 719. Die Anzahl der freien Sektoren ab 720 steht in Byte 122, 123 von Sektor 1024. Auch dies dient der Kompatibilität zu DOS 2.0s.

Byte 5-9: reserviert

Bytes 10-99 bzw. Bytes 10-137: Jedes Bit dieser Bytes repräsentiert die Belegung eines Sektors. Ist ein Bit 1, so ist der zugehörige Sektor frei, ist es jedoch 0, so ist der Sektor belegt. Die Zuordnung der Bits und Bytes zu den Sektoren geschieht wie folgt: Das am weitesten links stehende Bit von Byte 10, also das höchstwertige, entspricht Sektor 0 (gibt es nicht!), Bit 6 von Byte 10 entspricht Sektor 1, Bit 0 von Byte 10 entspricht Sektor 7, Bit 7 von Byte 11 gehört zu Sektor 8 usw.

2.12.4 Benutzung der DOS-Funktionen über die CIO

Allgemeine Informationen zur Funktionsweise der CIO finden sich in einem eigenen Abschnitt. Der folgende Abschnitt setzt das Lesen des CIO-Abschnitts voraus.

OPEN
Bei der Verwendung des OPEN-Kommandos sind folgende Werte des AUX1-Bytes möglich:

Dez	Hex	Funktion
12	$C	Sogenannter "Update"-Modus, bei dem sowohl gelesen als auch geschrieben werden kann.
9	$9	Schreiben ("Append"-Modus): Alle geschriebenen Bytes werden an das schon vorhandene File gehängt, d.h., das File wird um die neuen Daten verlängert.
8	$8	Schreiben.
6	$6	Lesen des Disketteninhaltsverzeichnisses ("Directory").
4	$4	Lesen.

Ein gültiger Name für das DOS-File ist wie folgt aufgebaut:

- Gerätename: D
- optional: Laufwerksnummer (1 bis 8); falls nicht angegeben, wird Laufwerk 1 angesprochen.
- Doppelpunkt :
- Name der Datei: bis zu acht Buchstaben oder Zahlen; unter DOS 2.0s darf das erste Zeichen keine Zahl sein.
- Punkt .
- Extender (Namenserweiterung): bis zu drei Buchstaben oder Zahlen
- EOL-Character (155;$9B)

CLOSE
Lesen: Der IOCB-Kanal wird zur weiteren Benutzung freigemacht.
Schreiben: Der Dateiname wird in das Directory eingetragen, sämtliche Puffer auf die Diskette geschrieben und die VTOC wird aktualisiert.

GET CHARACTERS und GET RECORD
Von der entsprechend geöffneten Datei wird die gewünschte Zahl der Zeichen eingelesen.

PUT CHARACTERS und PUT RECORD
Auf die entsprechend geöffnete Datei wird die gewünschte Zahl der Zeichen geschrieben.

STATUS
Der Status-Befehl kann zur Abfrage benutzt werden, ob eine Datei vorhanden und/oder nicht gesperrt ist. Dazu muss ICBADR auf die Dateispe-

zifikation zeigen und der verwendete Kanal muss geschlossen sein. Ist die Datei nicht vorhanden, erfolgt Fehler 170, ist sie gesperrt, Fehler 167.

SPECIAL
Folgende Kommandobefehle für die CIO, die alle einen Wert größer als 13 haben, werden als SPECIAL-Befehle betrachtet und speziell vom Diskettentreiber behandelt:

RENAME (Kommandonummer: 32, $20)
Mit diesem Befehl wird der Name einer Datei geändert. ICBADR muss dabei auf die Anfangsadresse der wie folgt aufgebauten Dateispezifikation zeigen:

- Gerätename: D
- optimal: Laufwerksnummer (1 bis 8), falls nicht angegeben, wird Laufwerk 1 angesprochen.
- Doppelpunkt :
- alter Name der Datei: bis zu acht Buchstaben oder Zahlen
- Punkt .
- alter Extender (Namenserweiterung): bis zu drei Buchstaben oder Zahlen
- Komma ,
- neuer Name der Datei: bis zu acht Buchstaben oder Zahlen
- Punkt .
- neuer Extender (Namenserweiterung): bis zu drei Buchstaben oder Zahlen
- EOL-Character (155;$9B)

DELETE (Kommandonummer: 33, $21)
Mit diesem Befehl wird die durch ICBADR adressierte Datei gelöscht.

LOCK (Kommandonummer: 35, $23)
Mit diesem Befehl wird die durch ICBADR adressierte Datei vor dem Löschen geschützt.

UNLOCK (Kommandonummer: 36, $24)
Mit diesem Befehl wird der Dateischutz der durch ICBADR adressierten Datei aufgehoben.

POINT (Kommandonummer: 37, $25)
Mit diesem Befehl wird die Lese-/Schreibposition der auf diesem Kanal geöffneten Datei neu gesetzt.

Für DOS-2-kompatible DOS gilt dabei: Es dürfen nur die mit NOTE zuvor ermittelten Werte benutzt werden und auch nur dann, wenn die betref-

fende Datei nicht umkopiert wurde. In AUX3 und AUX4 muss die neue Sektornummer, in AUX5 die Bytenummer innerhalb dieses Sektors eingetragen werden.

Bei SD-kompatiblen DOS wird mit POINT die Lese-/Schreibposition als Abstand vom Dateianfang in Bytes gesetzt, der in AUX3-5 anzugeben ist. Im SDFS können daher Dateien beliebig umkopiert werden.

NOTE (Kommandonummer: 38, $26)
Mit diesem Befehl kann die Lese-/Schreibposition der auf diesem Kanal geöffneten Datei festgestellt werden. Nach Rückkehr aus der CIO enthält AUX3-5 die aktuelle Lese-/Schreibposition in dem vom POINT-Befehl benötigten Format.

FORMAT SINGLE (Kommandonummer: 253, $FD)
Mit diesem Befehl wird unter DOS 2.5 eine Diskette im 720-Sektorformat initialisiert. ICBADR zeigt dabei auf eine Dateispezifikation, die lediglich die Gerätespezifikation des anzusprechenden Laufwerkes enthalten muss (beispielsweise "D1:").

FORMAT (Kommandonummer: 254, $FE)
Mit diesem Befehl wird unter DOS 2.0s eine Diskette im 720-Sektorenformat initialisiert. Unter DOS 2.5 hingegen wird die Diskette im 1040-Sektorenformat initialisiert. ICBADR zeigt dabei auf eine Dateispezifikation, die lediglich die Gerätespezifikation des anzusprechenden Laufwerkes enthalten muss (beispielsweise "D1:").

2.12.5 Fehlermeldungen von DOS 2.x

DOS-Fehlermeldungen sind abhängig vom verwendeten DOS und können nach Inhalt und Anzahl variieren. DOS 2.x meldet diese Fehler:

Error 128 - BREAK key abort
Es wurde während eines I/O-Vorgangs BREAK gedrückt.

Error 132 - Invalid command
Ein unbekanntes Ein- oder Ausgabekommando wurde verwendet.

Error 136 - End of file
Beim Lesen wurde das Dateiende erreicht.

Error 137 - Truncated record
Der Abstand zwischen zwei EOL-Zeichen in der Datei war so groß, dass sich der Record nicht in einem Stück lesen ließ.

Error 138 - Device timeout
Zum anzusprechenden Gerät besteht keine Verbindung!

Error 139 - Device NAK
Das Peripheriegerät kennt das auszuführende Kommando nicht. Z.B. der Versuch auf einem 810-Laufwerk in Medium Density zu formatieren.

Error 140 - Serial bus error
Bei Datenübertragung über den seriellen Bus kam es zu einem Fehler.

Error 142 - Serial bus data frame overrun
Bei Datenübertragung über den seriellen Bus kam es zu einem Fehler.

Error 143 - Serial bus data frame checksum error
Prüfsummenfehler bei der Datenübertragung über den seriellen Bus.

Error 144 - Device done error
Das Gerät konnte ein Kommando nicht ausführen. Die Fehlerursache dafür liegt beim Gerät bzw. Datenträger, nicht beim Computer. Z.B. konnte die Diskette nicht beschrieben werden, weil sie schreibgeschützt ist, oder es tauchte bei dem Versuch, einen bestimmten Sektor zu lesen, ein Fehler auf.

Error 146 - Function not implemented
Es wurde versucht, einem Gerät ein CIO-Kommando zu geben, das der betreffende Gerätetreiber nicht unterstützt.

Error 160 - Drive number error
Das angesprochene Laufwerk ist im DOS nicht angemeldet. Unter ATARI DOS 2.x ist dazu Speicherstelle DRVBIT (1802; $70A) anzupassen.

Error 161 - Too many open files
Es wurde versucht, mehr Dateien zu öffnen, als es möglich ist. Unter ATARI DOS 2.x ist für mehr offene Dateien Speicherstelle SABYTE (1801; $709) anzupassen.

Error 162 - Disk full
Beim Schreiben auf Diskette wurde festgestellt, dass kein freier Sektor mehr vorhanden ist.

Error 163 - Disk incompatible
Die eingelegte Diskette ist nicht DOS-kompatibel, d.h. hat keine korrekte VTOC. (Wenn dieser Fehler nicht auftritt, bedeutet das allerdings nicht, dass die Diskette wirklich DOS-kompatibel ist.) Bei ATARI DOS 2.x tritt dieser Fehler genau dann auf, wenn Byte 5 von Sektor 360 nicht 0 ist.

Error 164 - File number mismatch
Es wurde versucht, einen Sektor zu lesen oder zu schreiben, der nicht zu der benutzten Datei gehört.

Error 165 - File name error
Es wurde ein ungültiger Dateiname verwendet.

Error 166 - POINT data length error
Die Bytenummer (AUX5) beim POINT-Befehl war zu groß.

Error 167 - File locked
Es wurde versucht, eine geschützte Datei zu beschreiben, zu löschen oder umzubenennen.

Error 168 - Command invalid
Es wurde versucht, ein dem Diskettentreiber unbekanntes Kommando oder den OPEN-Befehl mit unbekannten Hilfsbytes zu benutzen. Dieser Fehler hat für den Nutzer praktisch dieselbe Bedeutung wie Error 146.

Error 169 - Directory Full
Die Höchstzahl von 64 Dateien auf der Diskette ist erreicht!

Error 170 - File not found
Die angegebene Datei existiert auf der Diskette nicht.

Error 171 - POINT invalid
Die Datei wurde in einem Modus geöffnet, in dem der ausgeführte POINT-Befehl nicht erlaubt ist. Bei DOS 2.x funktioniert POINT nur bei Dateien, die zum Lesen oder Ändern (Update) geöffnet wurden.

Error 173 - Bad sector at format
Beim Formatieren hat das Diskettenlaufwerk defekte Sektoren gefunden, die Diskette ist unbrauchbar. Dieser Fehler taucht erst nach mehreren Minuten auf, da das Laufwerk mehrere Formatierversuche unternimmt. Manche DOS-Versionen wie z.B. MyDOS liefern stattdessen Error 144.

2.12.6 Zahl der Diskettenlaufwerke (DOS 2.x)

Die Zahl der angeschlossenen Diskettenlaufwerke kann im Register DRVBYT (1802; $70A) eingestellt werden. Jedes gesetzte Bit dieses Bytes repräsentiert ein Laufwerk. Dabei steht Bit 0 für Laufwerk 1, Bit 1 für Laufwerk 2 usw. Eine Anpassung dieses Registers an die tatsächlich vorhandene Gerätekonfiguration ist insofern von Vorteil, als einerseits Speicherplatz gespart und andererseits die Länge des Bootvorgangs verkürzt wird, da nicht erst auf nicht existierende Laufwerke überprüft wird.

Die Zahl der Dateien, die gleichzeitig geöffnet sein können, ist im Register SABYTE (1801; $709) zu finden.

2.13 Editor

Der Editor (E:) ist ein Gerätetreiber, der als Eingabegerät Tastatur und Bildschirm und als Ausgabegerät den Bildschirm verwendet. Er ermöglicht das Editieren logischer Zeilen auf dem Bildschirm und in beschränktem Umfang eine formatierte Ausgabe von Daten. Folgende CIO-Funktionen werden von ihm unterstützt:

OPEN
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS
PUT RECORD
GET STATUS (Dummy-Funktion)

Der Editor kann Ein- und Ausgabefunktionen in drei Hauptkombinationen ausführen:

- Daten von der Tastatur lesen und gleichzeitig auf dem Bildschirm darstellen.

- Vom aufrufenden Programm gelieferte Daten auf dem Bildschirm darstellen.

- Daten vom Bildschirm lesen und dem aufrufenden Programm zugänglich machen.

Da der Editor in erster Linie Teile des Tastaturtreibers und des Bildschirmtreibers verwendet, wird hier auf die Beschreibung von Fakten verzichtet, die sich aus den Eigenschaften dieser beiden Gerätetreiber ergeben.

Kernpunkt des Editors ist, dass der Anwender eine vollständige Kontrolle darüber hat, welche Bildschirmdaten dem aufrufendem Programm zugänglich gemacht werden sollen: Zunächst können beliebig lange logischen Zeilen auf dem Bildschirm auf verschiedene Art und Weise editiert werden. Erst wenn die RETURN-Taste gedrückt wird, übergibt der Editor die Daten der logischen Zeile, in der der Cursor zuletzt stand, Byte für Byte an das aufrufende Programm. Am Anfang dieser logischen Zeile stehende Leerzeichen werden dabei entfernt. Schließlich wird der Cursor auf den Anfang der folgenden logischen Zeile gesetzt.

Im Folgenden eine Beschreibung der einzelnen CIO-Funktionen:

OPEN
Gerätespezifikation ist "E:"; Gerätenummern und Dateinamen werden ignoriert. Über das AUX1-Register können folgende verschiedene Modi eingestellt werden:
Bit 3 gesetzt: Schreiben
Bit 2 gesetzt: Lesen
Bit 0 gesetzt: Automatisches Lesen vom Bildschirm

CLOSE
Der Editor führt hierbei keine weiteren Funktionen aus; es wird lediglich der belegte Kanal für eine neue Anwendung geräumt.

GET CHARACTERS und GET RECORD
Diese Funktion unterscheidet sich von den GET-Routinen anderer Gerätetreiber dadurch, dass Daten stets zeilenweise (nach Drücken von RETURN) gelesen und dann Byte für Byte übergeben werden. So kann es einerseits passieren, dass beim Lesen eines einzelnen Bytes vom Editor zunächst vom Benutzer eine komplette Zeile eingetippt oder verändert wird, bevor das erste Zeichen dieser Zeile als Ergebnis zurückgeliefert wird. Andererseits kann es auch vorkommen, dass der Editor gar nicht erst neue Zeichen von der Tastatur einliest, da die letzte Zeile noch nicht komplett an das aufrufende Programm übergeben wurde.

Ein Sonderfall ergibt sich, wenn der Cursor zu Beginn nicht am Anfang der logischen Zeile steht - beispielsweise weil die logische Zeile mit einer Eingabeaufforderung (wie z. B. "Name?") beginnt. In einem solchen Fall wird nur dann die gesamte logische Zeile an den Editor übergeben, wenn der Cursor während des Editierens die logische Zeile verlassen hat. Ansonsten wird die logische Zeile nur ab der ursprünglichen Startposition des Cursors übergeben.

Ein weiterer Sonderfall ist der Modus mit automatischem Einlesen (Forced Read). In dieser Betriebsart "drückt" der Editor die RETURN-Taste sozusagen automatisch, d.h., er liest die logische Zeile, in der der Cursor steht, ohne erst darauf zu warten, dass über die Tastatur editiert wird.

Nach Ausführung des RETURN wird der Cursor auf den Anfang der nächsten logischen Zeile gesetzt.

PUT BYTES und PUT RECORD
Die Funktion der PUT-Routine des Editors entspricht, von den nachfolgend angegebenen Sonderfällen abgesehen, der PUT-Routine des Bildschirmtreibers in Grafikmodus 0.

Alle Zeichen werden im ATASCII-Code ausgegeben, sofern es sich nicht um eine der folgenden Editierfunktionen handelt:

ESCAPE (27; $1B): das nächste Zeichen, sofern nicht EOL (155; $9B), wird, falls es sich um eine dieser Steuerfunktionen handelt, als normales Zeichen auf dem Bildschirm ausgegeben. So ist es auch möglich, die Zeichen, die mit Editierfunktionen belegt sind, auf den Bildschirm zu bringen. Dazu drückt man zunächst einmal ESCAPE und dann die Tastenkombination, mit der normalerweise die Editierfunktion aufgerufen würde. Um einen Pfeil nach links auf dem Bildschirm auszugeben, müsste man also zuerst ESCAPE und daraufhin CONTROL+ drücken.

Cursor hoch (28; $1C): Der Cursor wird um eine Zeile nach oben bewegt. Befand sich der Cursor bereits in der obersten Zeile, erscheint er danach in der untersten Zeile.

Cursor runter (29; $1D): Der Cursor wird um eine Zeile herunter bewegt. Befand sich der Cursor bereits in der untersten Zeile, erscheint er danach in der obersten Zeile.

Cursor nach links (30; $1E): Der Cursor wird um eine Spalte nach links bewegt. Befand sich der Cursor bereits in der am weitesten links stehenden Spalte, erscheint er danach am rechten Bildschirmrand.

Cursor nach rechts (31; $1D): Der Cursor wird um eine Spalte nach rechts bewegt. Befand sich der Cursor bereits in der am weitesten rechts stehenden Spalte, erscheint er danach am linken Bildschirmrand.

Löschen (125; $7D): Der gesamte Textbildschirm wird gelöscht und der Cursor in die linke obere Ecke gesetzt.

Zeichen zurück (126; $7E): Der Cursor wird um ein Zeichen zurückbewegt und daraufhin das Zeichen unter dem Cursor gelöscht. Befand sich der Cursor bereits im ersten Zeichen der logischen Zeile, passiert nichts.

Tabulator (127; $7F): Der Cursor wird an die nächste Position innerhalb der logischen Zeile gesetzt, an der ein Tabulatorstopp gesetzt ist. Wenn innerhalb der augenblicklichen Länge der logischen Zeile kein Tabulatorstopp mehr vorliegt, wird der Cursor auf den Beginn der nächsten logischen Zeile gesetzt.

Zeilenende (EOL, 155; $9B): Die aktuelle logische Zeile wird an den Editor übergeben und der Cursor auf den Beginn der nächsten logischen Zeile gesetzt.

Zeile löschen (156; $9C): Die logische Zeile, in der sich der Cursor befindet, wird gelöscht. Alle darunter stehenden logischen Zeilen werden nach oben bewegt und schließen die Lücke. Am unteren Bildschirmrand werden leere logische Zeilen erzeugt.

Zeile einfügen (157; $9D): Alle physikalischen Zeilen unter der momentanen Cursorposition werden um eine Zeile nach unten bewegt. An der aktuellen Cursorposition wird eine leere physikalische Zeile eingesetzt, die gleichzeitig Beginn einer neuen logischen Zeile wird.

Tabulator zurücksetzen (158; $9E): Falls an der momentanen Position in der logischen Zeile ein Tabulatorstopp gesetzt ist, wird dieser gelöscht.

Tabulator setzen (159; $9F): An der momentanen Position der logischen Zeile wird ein Tabulatorstopp gesetzt.

Summer (253; $FD): Es wird ein Brummton ausgegeben. Beim ATARI 400 und 800 erfolgte dies über den eingebauten Lautsprecher, bei den neueren Geräten über den Lautsprecher im Fernseher.

Zeichen löschen (254; $FE): Das Zeichen unter dem Cursor wird gelöscht; alle nachfolgenden Zeichen der logischen Zeile werden um eine Position nach links verschoben.

Zeichen einfügen (255; $FF): Das Zeichen unter dem Cursor und alle nachfolgenden Zeichen werden um eine Position nach rechts verschoben und an der Cursorposition ein Leerzeichen eingefügt.

GET STATUS
Das Ergebnis dieser Funktion ist stets 1 (Status OK).

Ein- und Ausgabe
Ein- und Ausgabevorgänge können vom Benutzer in verschiedenen Ebenen beeinflusst werden. Dabei sollte jedoch, falls irgendwie möglich, der vollständige und exakte Weg über die CIO (Central I/O Utility) genommen werden.

Die CIO bildet beim Zugriff auf Peripheriegeräte die oberste und leistungsfähigste Ebene.

Obwohl für die einzelnen Operationen der CIO Tabellen mit Vektoren für jedes einzelne Gerät vorliegen, sollte beachtet werden, dass die korrekte Funktion auf späteren Betriebssystemversionen nicht unbedingt gewährleistet ist! Schließlich kann bei der Ein- und Ausgabe auch der direkte Weg genommen werden. Bei Ausgaben auf den Bildschirm hieße das bei-

spielsweise, dass die Daten direkt in den Bildspeicher geschrieben werden müssen. Dies ist übrigens dann notwendig, wenn die Grafikstufe, in der gearbeitet werden soll, nicht vom Betriebssystem unterstützt wird.

Für externe Peripherie, wie Drucker, Kassettenrecorder oder Diskettenstation, gibt es auch die Möglichkeit, die Routinen der SIO (→ SIO) zu nutzen oder gar direkt die Hardwareregister für die serielle Schnittstelle zu verwenden.

2.14 Fließkomma-Arithmetik

Im Betriebssystem sind einige Routinen zur Verwendung von Fließkommazahlen enthalten. Dies hat den Vorteil, dass alle Programmiersprachen und auch einige Programme problemlos auf die Routinen zugreifen können.

Da es leider keine Einsprungtabelle für die einzelnen Programme gibt, muss man die Routinen an ihrer absoluten Position im ROM aufrufen. Da die Fließkommaroutinen durch ATARI für den eigenen Gebrauch "freigegeben" worden sind und sich die Anfangsadressen in den verschiedenen Betriebssystemversionen sich bisher tatsächlich nicht geändert haben, sollten sich dabei eigentlich auch bei künftigen Veränderungen des Betriebssystems keine Probleme ergeben.

Fließkommazahlen werden jeweils innerhalb von sechs Bytes abgespeichert. Dabei wird das erste Byte für Exponent und Vorzeichen, alle übrigen Bytes für die Mantisse benutzt.

Für das Vorzeichen der Zahl ist Bit 7 des ersten Bytes reserviert. Es hat für negative Zahlen den Wert 1, für positive den Wert 0. Die übrigen sieben Bits, die noch Werte zwischen 0 und 127 einnehmen können, repräsentieren den Exponenten. Beachtet werden muss, dass es sich dabei um einen Exponenten zur Basis 100 handelt. Zum Wert des Exponenten muss 64 addiert werden, um die unteren sieben Bits zu errechnen - z.B. entspricht dem Exponenten 0 der 7-Bit-Wert 64 und dem Exponenten -1 der Wert 63.

Die restlichen fünf Bytes enthalten die Mantisse der Zahl im BCD-Format. Das heißt, dass eine Ziffer jeweils durch einen 4-Bit-Wert dargestellt wird und so in jedem Byte zwei Ziffern abgespeichert sind. Da als Basis die Zahl 100 verwendet wird, befindet sich der Dezimalpunkt stets hinter dem ersten Byte der Mantisse (also nach der zweiten Ziffer).

Eine Sonderstellung nimmt die Zahl 0 ein, die einfach durch sechs Nullen (000000) dargestellt wird.

Das Fließkommapaket ("Floating Point Package") stellt folgende Routinen zur Bearbeitung von Zahlen im Fließkommaformat (F.P.) zur Verfügung:

- Umwandlung einer Zahl im ASCII-Format in Fließkommaformat.
- Umwandlung einer Zahl im Fließkommaformat in ASCII-Format.
- Umwandlung einer 2-Byte-Integer-Zahl in Fließkommaformat.
- Umwandlung einer Fließkommazahl in 2-Byte-Integer-Format.
- Addition, Subtraktion, Multiplikation und Division von Fließkommazahlen.
- Exponentialrechnung und Berechnung von Logarithmen und Polynomen.
- Verschieben, Löschen, Kopieren und Abspeichern von Fließkommawerten.

Folgende Register werden von den Fließkommaroutinen benutzt:

FR0 (212-217; $D4-$D9):	Register für Fließkommazahlen
FR1 (224-229; $E0-$E5):	Register für Fließkommazahlen
CIX (242; $F2):	Index in den Puffer, auf den INBUFF zeigt
INBUFF (243,244; $F3,$F4,):	Zeiger auf Zahl im ASCII-Format
FLPTR (252,253; $FC,$FD):	Zeiger auf eine Fließkommazahl
LBUFF (1408; $580):	Ergebnisspeicher der FASC-Routine

Insgesamt verwendet das Fließkommapaket alle Register zwischen 212 ($D4) und 255 ($FF), sowie den Bereich von 1406 ($57E) bis 1535 ($5FF).

Beachtet werden muss außerdem, dass die Treiberroutinen für Geräte, die am parallelen Bus angeschlossen sind, keine Fließkommaroutinen verwenden dürfen, da dieses ja in einem solchen Fall ausgeblendet wird und eben diesen Routinen Platz macht!

Die Benutzung der einzelnen Register hängt jeweils von der betreffenden Routine ab. Das Übertrags-Flag (Carry-Bit) wird jedoch zumeist für Fehlermeldungen verwendet.

Nun zur Besprechung der einzelnen Fließkommaroutinen:

AFP — Umwandlung ASCII → FP
Mit dieser Routine kann man eine Zahl im ASCII-Format in Fließkommaformat umwandeln.

Benötigte Register:
INBUFF → Zeiger auf Speicherbereich mit Zahl im ASCII-Format
CIX → Index innerhalb dieses Puffers zum ersten Zeichen der Zahl

AFP (55296; $D800) liest so lange Zeichen aus dem angegebenen Puffer, bis ein Zeichen auftritt, das nicht umgewandelt werden kann. Gültige Zeichen sind die Ziffern, "+", "-", "." und "E". Anschließend wird die Zahl in Fließkommaformat umgewandelt.

Veränderte Register:
Ein gesetztes Carry-Flag signalisiert, dass keine gültige Fließkommazahl gefunden werden konnte.
FR0 enthält die umgewandelte Zahl im Fließkommaformat.
CIX zeigt nun auf das erste Byte nach der umgewandelten Zahl im ASCII-Format (sehr praktisch bei der Verarbeitung von Zahlenlisten).

FASC — Umwandlung FP → ASCII
Diese Routine wandelt eine Zahl im Fließkommaformat in ASCII-Format.

Benötigte Register:
FR0 → Zu wandelnde Fließkommazahl

FASC (55526; $D8E6) verwandelt die Fließkommazahl in FR0 in eine Zahl im ASCII-Format.

Veränderte Register:
INBUFF → zeigt auf die verwandelte Zahl im ASCII-Format. Das Ende der Zahl wird durch ein invertiertes, letztes Zeichen gekennzeichnet.

IFP — Umwandlung Integer → FP
Mit dieser Routine kann eine 2-Byte-Integer-Zahl in eine Fließkommazahl verwandelt werden.

Benötigte Register:
FR0 → Low Byte des Integers
FR0+1 → High Byte des Integers

IFP (55722; $D9AA) wandelt den Integerwert in FR0 in eine Fließkommazahl um.

Veränderte Register:
FR0 → Enthält nun die Zahl im Fließkommaformat.

FPI — Umwandlung FP → Integer
Diese Routine verwandelt Fließkommazahlen in 2-Byte-Integer-Werte.

Benötigte Register:
FR0 → umzuwandelnde Fließkommazahl

FPI (55762; $D9D2) wandelt eine Fließkommazahl in einen Integerwert um. Dabei wird die Fließkommazahl, falls nötig, gerundet.

Veränderte Register:
Das Carry-Flag ist gesetzt, wenn die Fließkommazahl außerhalb des durch 2-Byte-Integer-Zahlen darstellbaren Bereiches lag.
FR0 → Low Byte des Integer-Wertes
FR0+1 → High Byte des Integer-Wertes

ZFR0 — FR0 löschen
Diese Funktion setzt den Wert von FR0 auf 0.

Benötigte Register:
keine

ZFR0 (55876; $DA44) löscht das Register FR0.

Veränderte Register:
FR0 → wird "0"

ZF1 — Page-0-FP-Register löschen
Routine zum Löschen eines beliebigen Registers auf Seite 0 des Speichers (0-255).

Benötigte Register:
X → Adresse des zu löschenden Registers

ZF1 (55878; $DA46) löscht den Inhalt des Fließkommaregisters, dessen Adresse im X-Register angegeben ist.

Veränderte Register:
Das betreffende Fließkommaregister ist gelöscht.

FSUB — FP Subtraktion
Routine zur Subtraktion zweier Fließkommazahlen.

Benötigte Register:
FR0 → Minuend
FR1 → Subtrahend

FSUB (55904; $DA60) subtrahiert FR1 von FR0.

Veränderte Register:
Ein gesetztes Carry-Flag bedeutet, dass bei der Subtraktion der zulässige Zahlenbereich über- oder unterschritten worden ist.

FR0 → Differenz (FR0-FR1)
Der Inhalt von FR1 wird von der Routine verändert!

FADD — FP Addition
Routine zur Addition zweier FP-Zahlen.

Benötigte Register:
FR0 → erster Summand
FR1 → zweiter Summand

FADD (55910; $DA66) addiert die Fließkommazahlen in FR0 und FR1.

Veränderte Register:
Ein gesetztes Carry-Flag bedeutet, dass bei der Addition der erlaubte Zahlenbereich über- oder unterschritten worden ist.
FR0 → Summe von FR0 und FR1
Der Inhalt von FR1 wird von der Routine verändert!

FMUL — FP Multiplikation
Routine zur Multiplikation zweier Fließkommazahlen.

Benötigte Register:
FR0 → Multiplikator
FR1 → Multiplikand

FMUL (56027; $DADB) multipliziert die Fließkommazahlen in den Registern FR0 und FR1 miteinander.

Veränderte Register:
Ein gesetztes Carry-Flag bedeutet, dass bei der Multiplikation der erlaubte Zahlenbereich über- oder unterschritten worden ist.
FR0 → Produkt (FR0*FR1)
Der Inhalt von FR1 wird von der Routine verändert.

FDIV — FP Division
Routine zur Division zweier Fließkommazahlen.

Benötigte Register:
FR0 → Dividend
FR1 → Divisor

FDIV (56104; $DB28) teilt die in FR0 stehende Fließkommazahl durch den Inhalt von FR1.

Veränderte Register:
Ein gesetztes Carry-Flag bedeutet, dass bei der Division der erlaubte Zahlenbereich über- oder unterschritten worden ist oder dass versucht worden ist, eine Division durch 0 vorzunehmen.
FR0 → Quotient (FR0/FR1)
Der Inhalt von FR1 wird durch die Routine verändert!

PLYEVL — Berechnung von Polynomen
Diese Routine kann einen beliebigen Polynom der Form

$$X_n*Y^n+X_{n-1}*Y^{n-1}+ \ldots +X_1*Y+X^0$$

berechnen.

Benötigte Register:
Y → High Byte der Anfangsadresse der Koeffizientenliste in Fließkommaformat
X → Low Byte der Anfangsadresse der Koeffizientenliste in Fließkommaformat
A → Anzahl der Koeffizienten
FR0 → Wert Y in Fließkommaformat

PLYEVL (56640; $DD40) berechnet den Wert des angegebenen Polynoms.

Veränderte Register:
Ein gesetztes Carry-Flag zeigt an, dass während der Berechnung des Polynoms ein Fehler aufgetreten ist.
FR0 → Wert des Polynoms

FLD0R — FR0 Wert zuweisen
Mit dieser Routine kann FR0 der Wert einer beliebigen anderen Fließkommavariablen zugewiesen werden.

Benötigte Register:
Y → High Byte der Adresse der Fließkommazahl
X → Low Byte der Adresse der Fließkommazahl

FLD0R (56713; $DD89) weist FR0 den Wert der Fließkommazahl zu, auf die das X- und Y-Register zeigen.

Veränderte Register:
FR0 → Wert der adressierten Fließkommazahl
FLPTR → Adresse der Fließkommazahl

FLD0P — FR0 Wert zuweisen
Mit dieser Routine kann FR0 der Wert einer beliebigen anderen Fließkommavariablen zugewiesen werden.

Benötigte Register:
FLPTR → Adresse der Fließkommazahl

FLD0P (56717; $DD8D) weist FR0 den Wert der Fließkommazahl zu, auf die FLPTR zeigt.

Veränderte Register:
FR0 → Wert der adressierten Fließkommazahl

FLD1R — FR1 Wert zuweisen
Mit dieser Routine kann FR1 der Wert einer beliebigen anderen Fließkommavariablen zugewiesen werden.

Benötigte Register:
Y → High Byte der Adresse der Fließkommazahl
X → Low Byte der Adresse der Fließkommazahl

FLD1R (56728; $DD98) weist den Wert der Fließkommazahl zu, auf die das X- und Y-Register zeigen.

Veränderte Register:
FR1 → Wert der adressierten Fließkommazahl
FLPTR → Adresse der Fließkommazahl

FLD1P — FR1 Wert zuweisen
Mit dieser Routine kann FR1 der Wert einer beliebigen anderen Fließkommavariablen zugewiesen werden.

Benötigte Register:
FLPTR → Adresse der Fließkommazahl

FLD1P (56732; $DD9C) weist FR1 den Wert der Fließkommazahl zu, auf die FLPTR zeigt.

Veränderte Register:
FR1 → Wert der adressierten Fließkommazahl

FST0R — FR0 abspeichern
Mit dieser Routine kann der Inhalt von FR0 an einer beliebigen Stelle im Speicher abgelegt werden.

Benötigte Register:
Y → High Byte der gewünschten Adresse
X → Low Byte der gewünschten Adresse

FST0R (56743; $DDA7) überträgt den Inhalt von FR0 an die durch das Y- und das X-Register spezifizierte Adresse.

Veränderte Register:
Angegebene Adresse enthält nun Inhalt von FR0.

FST0P — FR0 abspeichern
Mit dieser Routine kann der Inhalt von FR0 an einer beliebigen Stelle im Speicher abgelegt werden.

Benötigte Register:
FLPTR → gewünschte Adresse

FST0P (56747; $DDAB) überträgt den Inhalt von FR0 an die durch FLPTR spezifizierte Adresse.

Veränderte Register:
Angegebene Adresse enthält nun Inhalt von FR0.

FMOVE — FR0 in FR1 übertragen
FMOVE (56758; $DDB6) überträgt den Inhalt von FR0 in FR1.

Veränderte Register:
FR1 → hat nun den Inhalt von FR0

EXP — Exponentialfunktion
Diese Routine berechnet die Funktion e^x.

Benötigte Register:
FR0 → Exponent

EXP (56768; $DDC0) berechnet mit dem Inhalt von FR0 als x den Wert der Formel e^x.

Veränderte Register:
Ein gesetztes Carry-Flag zeigt an, dass der zulässige Zahlenbereich verlassen worden ist.
FR0 → Funktionsergebnis
Der Inhalt von FR1 wird durch die Routine verändert!

EXP10 — Exponentialfunktion
Diese Routine berechnet die Funktion 10x.

Benötigte Register:
FR0 → Exponent

EXP10 (56780, $DDCC) berechnet mit dem Inhalt von FR0 als den Wert x der Formel 10^x.

Veränderte Register:
Ein gesetztes Carry-Flag zeigt an, dass der zulässige Zahlenbereich verlassen worden ist.
FR0 → Funktionsergebnis
Der Inhalt von FR1 wird durch die Routine verändert!

LOG — natürlicher Logarithmus
Routine zur Berechnung des natürlichen Logarithmus einer Fließkommazahl.

Benötigte Register:
FR0 → Fließkommazahl, deren natürlicher Logarithmus berechnet werden soll.

LOG (57037, $DECD) berechnet den natürlichen Logarithmus der in FR0 stehenden Fließkommazahl.

Veränderte Register:
Ein gesetztes Carry-Flag bedeutet, dass bei der Logarithmierung der erlaubte Zahlenbereich über- oder unterschritten worden ist und dass der Ausgangswert eine negative Zahl war.

FR0 → natürlicher Logarithmus des Ausgangswertes
Der Inhalt von FR1 wird von der Routine verändert!

LOG10 — Logarithmus zur Basis 10
Routine zur Berechnung des Logarithmus zur Basis 10 einer Fließkommazahl.

Benötigte Register:
FR0 → Fließkommazahl, deren Logarithmus zur Basis 10 berechnet werden soll.

LOG10 (57041; $DED1) berechnet den Logarithmus zur Basis 10 der in FR0 stehenden Fließkommazahl.

Veränderte Register:
Ein gesetztes Carry-Flag bedeutet, dass bei der Logarithmierung der erlaubte Zahlenbereich über- oder unterschritten worden ist oder dass der Ausgangswert eine negative Zahl war.
FR0 → Logarithmus zur Basis 10 des Ausgangswertes
Der Inhalt von FR1 wird von der Routine verändert!

Umrechnungsroutinen			
Name	Adresse		Beschreibung
	Dez	Hex	
AFP	55296	$D800	ASCII → Fließkomma
FASC	55526	$D8E6	Fließkomma → ASCII
IFP	55722	$D9AA	Integer → Fließkomma
FPI	55762	$D9D2	Fließkomma → Integer

Tab. 2.13: Übersicht der Fließkommaroutinen – Umrechnen

Speicherroutinen			
Name	Adresse		Beschreibung
	Dez	Hex	
ZFR0	55876	$DA44	FR0 löschen
ZF1	55878	$DA46	Page-0-Register löschen
FLD0R	56713	$DD89	(X,Y) → FR0
FLD0P	56717	$DD8D	(FLPTR) → FR0
FLD1R	56728	$DD98	(X,Y) → FR1
FLD1P	56732	$DD9C	(FLPTR) → FR1
FST0R	56743	$DDA7	FR0 → (X,Y)
FST0P	56747	$DDAB	FR0 → (FLPTR)
FMOVE	56758	$DDB6	FR0 → FR1

Tab. 2.14: Übersicht der Fließkommaroutinen – Speichern

Rechenroutinen			
Name	Adresse		Beschreibung
	Dez	Hex	
FSUB	55904	$DA60	FR0 -FR1 → FR0
FADD	55910	$DA66	FR0+FR1 → FR0
FMUL	56027	$DADB	FR0*FR1 → FR0
FDIV	56104	$DB28	FR0/FR1 → FR0
PLYEVL	56640	$DD40	Polynomberechnung
EXP	56768	$DDC0	e^{FR0} → FR0
EXP10	56780	$DDCC	10^{FR0} → FR0
LOG	57037	$DECD	ln(FR0) → FR0
LOG10	57041	$DED1	log10(FR0) → FR0

Tab. 2.15: Übersicht der Fließkommaroutinen – Rechnen

2.15 Gerätetreiber

Bei Ein- und Ausgabefunktionen liegt die Ebene der Gerätetreiber unmittelbar unter der CIO (→ CIO). Damit Anzahl und Art von angeschlossenen Geräten veränderbar sind, gibt es im RAM eine Tabelle, die die Geräteidentifikationszeichen und Zeiger auf die Vektortabellen der zu diesen Geräten gehörigen Routinen enthält. Diese Treibertabelle (Handler Table) heißt HATABS (794; $31A) und kann bis zu 11 Einträge umfassen. Jeder Eintrag besteht aus drei Bytes, wobei das erste Zeichen der ATASCII-Code des Gerätenamens und die beiden folgenden Bytes die Adresse der dazugehörigen Vektortabelle sind. Das letzte Byte dient als Endmarkierung und ist stets 0.

Beim Einschalten werden alle Treiber, die im Betriebssystem integriert sind, nämlich E, K, S, P und C, in die Treibertabelle eingetragen. Dabei wird von "unten" nach oben vorgegangen, sodass zunächst die Register 794 ($31A) bis 808 ($328) belegt werden.

(X) Danach wird nach PBI-Geräten gesucht, um das ROM vorhandener PBI-Geräte einzublenden und deren PDINIT-Routinen (55321; $D819) aufzurufen. Diese können in die Treibertabelle die Adresse der generischen Parallelbus-Vektortabelle GPDVV (58511; $E48F) eintragen, die die eigentlichen Vektortabellen der PBI-Geräte korrekt aufrufen kann.

Danach wird nach dem Laden des Diskettenbetriebssystems darüber noch der Eintrag für den Diskettentreiber (also das DOS) vorgenommen.

Bei einem OPEN-Befehl durchsucht die CIO die Treibertabelle nach dem angegebenen Gerätenamen, und zwar vom Ende zum Anfang hin. Bei Doppeleinträgen wird mithin der zuletzt gemachte Eintrag für gültig befunden.

Dieser "Unterbau" der CIO erlaubt es, ohne Probleme neue Gerätetreiber zu installieren und bestehende zu verändern. Diese "Bündelung" sämtlicher Ein- und Ausgabevorgänge über eine Einsprungadresse und die flexible Gestaltung der Treibertabelle machen es möglich, dass die Programme von heute prinzipiell mit der Hardware von morgen (80-Zeichen-Karten, parallele Diskettenlaufwerke etc.) kommunizieren können.

Ein anderes Beispiel: Achtet man darauf, dass keines Programmes Speicherbereichsgrenzen verletzt werden, könnte man den gesamten Editor umschreiben - alle Programme, die Ein-/Ausgabe ausschließlich über die CIO betreiben, werden auch hiermit problemlos funktionieren!

2.15.1 Aufbau der Treibertabelle

Schauen wir uns doch mal den ersten Eintrag in der Treibertabelle an (könnte bei einem anderen Betriebssystem als dem vom XL/XE auch ein anderer Eintrag sein!):

$31A = 794: $50
$31B = 795: $30
$31C = 796: $E4

Das erste Byte hat den Wert $50, was nicht zufällig dem ATASCII-Wert des Buchstabens P entspricht: Dies ist also der Eintrag für den Druckertreiber. Die nächsten beiden Bytes sind mithin die Adresse der Vektortabelle, die wir uns gleich etwas näher ansehen wollen. Wie fast ausschließlich beim ATARI, steht auch hier das niederwertige Byte an erster Stelle, die gesuchte Vektortabelle PRINTV findet man also bei der Adresse 58416 ($E430). In diesem Bereich finden sich übrigens auch alle anderen Vektortabellen für die durch das Betriebssystem unterstützten Treiber:

EDITRV:	$E400 = 58368	Editor
SCRENV:	$E410 = 58384	Bildschirm
KEYBDV:	$E420 = 58400	Tastatur
PRINTV:	$E430 = 58416	Drucker
CASETV:	$E440 = 58432	Kassette
GPDVV:	$E48F = 58511	Paralleler Bus

Inzwischen ist sicherlich die Frage aufgetaucht, warum denn der Treiber für den Diskettenzugriff, also das Diskettenbetriebssystem (DOS), nicht auch im Betriebssystem liegt. Hierfür kann man zwei Gründe nennen: Einerseits ist das natürlich auch eine Frage des Platzes, denn ein DOS ist ein wenig komplizierter als ein "normaler" Gerätetreiber. Andererseits ist so die Möglichkeit geblieben, mit völlig verschiedenen DOS-Versionen zu arbeiten. Man denke dabei an die vielen Spezialversionen für eine Vielzahl von Diskettenlaufwerken von Fremdanbietern - beispielsweise mit doppelter Schreibdichte!

Adresse		ATASCII	Vektor	
$031A		P	$E430	➔ Tab. 2.17: Beispiel Druckertreiber
$031D		C	$E440	
$0320		E	$E400	
$0323		S	$E410	
$0326		K	$E440	
$0329		D	$07CB (Beispiel für DOS 2.0)	
$032C		...	...	
...		...	...	

🡅 bis zu 5 weitere Treiber möglich

Tab. 2.16: Treibertabelle mit den Gerätetreibereinträgen

2.15.2 Aufbau einer Vektortabelle

Aus der Übersicht über die integrierten Vektortabellen konnte man schon ohne Weiteres darauf schließen, dass jede Tabelle genau 16 Bytes lang ist. Wie ist sie nun aufgebaut? Das Schema in Tab. 2.17 ist folgendermaßen zu verstehen: Die ersten sechs Doppelbytes sind die Vektoren zu den entsprechenden Unterprogrammen, wobei man darauf achten muss, dass jeweils der Vektor auf das Byte vor der eigentlichen Routine zeigt. Das hat folgenden Grund: Indem man das höherwertige und das niederwertige Byte auf den Prozessor-Stack schiebt und einen RTS-Befehl durchführt, kann man die gewünschte Routine aufrufen. Der Grund für dieses Vorgehen liegt in der internen Arbeitsweise der CIO. Darauf folgt ein Maschinensprache-Sprungbefehl (JMP, $4C) zu einem Unterprogramm, das das betreffende Gerät initialisiert.

Beispiel: Druckertreiber	
Adresse	
$E430	Vektor zur OPEN-Routine (-1)
$E432	Vektor zur CLOSE-Routine (-1)
$E434	Vektor zur GET BYTE-Routine (-1)
$E436	Vektor zur PUT BYTE-Routine (-1)
$E438	Vektor zur STATUS-Routine (-1)
$E43A	Vektor zur SPECIAL-Routine (-1)
$E43C	JMP ($4C)
$E43D	Adresse der INIT-Routine
$E43F	unbenutzt

Tab. 2.17: Vektortabelle eines Gerätetreibers

Außer bei den im ROM vorhandenen Vektortabellen der Geräte P, E, S, K und C wird dieser Sprungbefehl vom Betriebssystem ignoriert. Daher müssen nachgeladene Treiber (z.B. der Diskettentreiber) über eine resetfeste Routine selbst dafür sorgen, dass beim Reset seine Initialisierungsroutine aufgerufen wird. Üblicherweise geschieht dies beim Laden des Treibers durch Verbiegen des DOSINI-Vektors (12,13; $C,$D): Man liest den alten DOSINI-Wert aus, setzt DOSINI dann auf die eigene Init-Routine, in der als erstes die alte DOSINI-Routine per JSR gerufen wird. Das machen auch andere Reset-feste Programme wie z.B. Turbo BASIC XL so (manchmal wird statt DOSINI auch CASINI (2,3; $2,$3) benutzt). Das letzte Byte der Vektortabellen ist (bisher noch) unbenutzt.

Die Initialisierungsroutine sollte folgende Aufgaben erfüllen:

Interne Zeiger, Puffer usw. initialisieren.
Peripheriegerät initialisieren.

Alles, was außerdem getan werden muss, um einen neuen Gerätetreiber zu entwickeln, den man über die CIO wie jeden Treiber auch aufrufen kann, ist das Schreiben von sechs Unterprogrammen, die die zuvor aufgeführten Funktionen ausführen.

Parameter werden von der CIO folgendermaßen übergeben:

Das X-Register enthält den Index zum aufrufenden IOCB - dies ist exakt derselbe Wert, den man auch vor einem Sprung nach CIOV in das X-Register lädt.

Das Y-Register enthält den Wert 146 ($92), was dem Fehlerwert für "Funktion nicht unterstützt" entspricht. Daher braucht man den Vektor nicht unterstützter Funktionen nur auf einen RTS-Befehl zeigen zu lassen.

Außerdem werden die ersten 12 Bytes des IOCBs, von dem aus der Treiber aufgerufen worden ist, in den Page-0-IOCB ab 32 ($20) kopiert. Dies geschieht aus zweierlei Gründen: Einerseits werden einige Register von der CIO während der Ausführung der Funktion verändert, andererseits erleichtert dies die Arbeit der eigenen Treiberprogramme.

Nach Beendigung der Funktion muss das Y-Register mit dem Statuswert der Operation geladen werden - hierbei entspricht "Status OK" dem Wert 1. Fehlerwerte sind stets größer als 127 und sollten möglichst weitgehend den üblichen Fehlermeldungen (siehe Anhang) entsprechen.

Zusätzlich zu übergebende Parameter werden bei der jeweiligen Treiberroutine erläutert. Diese Ein- und Ausgabeparameter sind für alle Funktionen gleich und werden daher nicht noch einmal extra erwähnt.

Kommen wir zu den einzelnen Routinen:

OPEN
Diese Routine wird von der CIO angesprochen, wenn ein direkter OPEN-Befehl an die CIO gegangen ist.

Dabei stellt die CIO zusätzlich folgende Parameter zur Verfügung:

ICDNOZ (33; $21): Gerätenummer
ICDBALZ/ICBAHZ (34,35; $22,$23): Adresse der Dateispezifikation
ICAX1Z/ICAX2Z (42,43; $2A/$2B): Vom Gerät abhängige Zusatzinformation; anhand dieser Parameter muss die OPEN-Routine das entsprechende Gerät initialisieren.

CLOSE
Diese Routine sollte möglicherweise noch in Zwischenspeichern stehende Bytes absenden, das Dateiende markieren und, falls notwendig, Belegungstabellen, Inhaltsverzeichnisse und Ähnliches aktualisieren.

Die CIO wird den betreffenden IOCB übrigens auch dann freigeben, wenn im Y-Register eine Fehlermeldung übertragen wurde.

GETBYTE
Die CIO ruft diese Routine infolge eines "GET CHARACTERS"- oder "GET RECORD"-Befehls auf. War zu diesem Zeitpunkt der IOCB noch nicht ge-

öffnet, dann wird GET BYTE gar nicht erst aufgerufen, sondern die CIO sorgt selbständig für eine passende Fehlermeldung.

Die "GET BYTE“-Routine soll ein einzelnes Byte entweder direkt vom betreffenden Gerät oder auch aus einem bereits bestehenden Zwischenspeicher lesen. Beim Rücksprung muss der Akkumulator dieses Byte enthalten.

Gerätetreiber, bei deren Leseoperationen signifikante Wartezeiten entstehen können, müssen selbständig den Status der BREAK-Taste überwachen und sollten in dem Fall, dass sie gedrückt ist, eine Fehlermeldung (meist $80) ausgeben!

PUTBYTE

Die CIO ruft diese Routine infolge eines "PUT CHARACTERS"- oder "PUT RECORD"-Befehls auf. War zu diesem Zeitpunkt der IOCB noch nicht geöffnet, dann wird PUT BYTE gar nicht erst aufgerufen, sondern die CIO sorgt selbständig für eine passende Fehlermeldung.

Zusätzlich zu den Standardparametern wird dieser Routine das zu übertragende Byte im Akkumulator übergeben.

PUTBYTE sollte das übergebene Byte entweder sofort zum jeweiligen Gerät übertragen oder aber es in einen selbst kontrollierten Zwischenspeicher schreiben, der dann periodisch (also blockweise) auf das betreffende Gerät übertragen werden kann.

Für die Arbeitsweise der "PUT BYTE"-Routine muss man Folgendes beachten, wenn sie einwandfrei mit ATARI BASIC oder einem beliebigen anderen Programm, das (verbotenerweise) den ICPUT-Vektor im IOCB benutzt, zusammenarbeiten soll: In diesem Fall dürfen Parameter nicht aus dem Page-0-IOCB gelesen werden, sondern sie müssen aus dem "echten" IOCB gelesen werden (im X-Register liegt ja der Zeiger auf den aufrufenden IOCB!). Außerdem muss man als zusätzlich möglichen Fehler den Aufruf der Routine ohne vorheriges Öffnen des IOCB abfangen.

GETSTAT

Die CIO ruft diese Routine infolge eines "GET STATUS"-Befehls auf. Aufgrund der "Implied Open"-Funktion (→ CIO) ist es möglich, dass bei Aufruf dieser Routine der betreffende Kanal noch nicht geöffnet ist. Falls dies irgendeine Auswirkung auf die GETSTAT-Funktion hat, dann muss sich diese selbständig um die möglichen Auswirkungen kümmern.

Bei SIO-Geräten sollte GETSTAT außerdem vier Bytes mit Statusinformationen in die vier Register ab DVSTAT (746; $2EA) übertragen.

SPECIAL
Die CIO ruft diese Routine immer dann auf, wenn ICCOM einen Wert enthält, der größer als 13 ($D) ist. Beispiele hierfür sind die Routinen DRAW, RENAME, DELETE etc.

Aufgrund der "Implied Open"-Funktion (→ CIO) ist es möglich, dass bei Aufruf dieser Routine der betreffende Kanal noch nicht geöffnet ist. Falls dies irgendeine Auswirkung auf die SPECIAL-Funktion hat, dann muss sich diese selbständig um die möglichen Auswirkungen kümmern.

SPECIAL muss anhand von ICCOMZ selbständig zwischen allen gerätetreiberspezifischen Funktionen unterscheiden und diese abhängig vom tatsächlichen Kommandobyte aufrufen.

Fehlerbehandlung
Die Behandlung von Fehlern ist insofern sehr einfach, als die CIO praktisch alle logischen Fehler abfängt. Bei Gerätetreibern, die die serielle Schnittstelle verwenden (→ SIO), kommt hinzu, dass ebenfalls alle Übertragungsfehler von der SIO bemerkt und zurückgemeldet werden. Der jeweilige Gerätetreiber braucht sich eigentlich nur um folgende Ereignisse zu kümmern:

Dateiende
Möglicherweise Drücken der BREAK-Taste
Erkennung unbekannter Befehle

Fehlermeldungen sollten ausschließlich als Wert im Y-Register des Prozessors an die CIO zurückgemeldet werden!

2.16 Hardware-Register

Unter Hardware-Registern versteht man die Register, die im Bereich eines Ein-/Ausgabechips liegen. Dabei handelt es sich um folgende Speicherbereiche:

$D000-$D0FF:	GTIA-Chip
$D100-$D1FF:	Steuerung des parallelen Bus
$D200-$D2FF:	POKEY-Chip
$D300-$D3FF:	PIA-Chip
$D400-$D4FF:	ANTIC-Chip
$D500-$D5FF:	CARCTL
$D600-$D6FF:	(noch) unbenutzt
$D700-$D7FF:	durch ATARI für Testzwecke bei der Softwareentwicklung reserviert

Hardwareregister haben zumeist eine unterschiedliche Lese- und Schreibfunktion. Beim Lesen von Hardwareregistern erhält man zumeist eine Statusmeldung des betreffenden Chips. Beim Schreiben in Hardwareregistern wird entweder eine bestimmte Funktion ausgelöst oder aber ein interner Status des Chips verändert (→ Schattenregister, Vertikal-Blank-Interrupt).

2.17 Joysticks und Paddles

Der ATARI 400 und 800 haben vier, die XL/XE-Geräte zwei Anschlüsse für Joysticks (Steuerknüppel). An jeden dieser Anschlüsse kann stattdessen auch ein Paar von Drehreglern (Paddles), eine Maus, eine Maltafel (Touch-Tablet), ein Lichtgriffel oder eine Lichtpistole angeschlossen werden. Darüber hinaus lassen sich diese Ports auch für andere Zwecke benutzten, da sie nicht nur Dateneingaben, sondern auch Datenausgaben ermöglichen. Hier ist vor allem die XEP80 zu nennen (→ XEP80).

Bei der Abfrage dieser Anschlüsse stehen zwei Ebenen zur Verfügung: Einerseits die Hardwareregister im Bereich des PIA (54016-54019; $D300-$D303), andererseits die durch das Betriebssystem bereitgestellten Variablen zwischen 624 ($270) und 647 ($287). Die Letzteren sind im Allgemeinen vorzuziehen, da einerseits einige lästige Umrechnungsarbeit eingespart wird und andererseits das Betriebssystem automatisch für die bei XL/XE-Geräten fehlenden Anschlüsse vernünftige Werte einsetzt (es kopiert nämlich einfach die Werte für die ersten beiden Anschlüsse in die Register für die fehlenden Anschlüsse). Außerdem ist für die Abfrage der PADDLE-Register im POKEY ein spezielles Timing erforderlich, auf das man so nicht zu achten braucht.

Die Bearbeitung der Drehreglerwerte ist denkbar einfach: Die Register enthalten stets einen Wert zwischen 0 (rechter Anschlag) und 228 (linker Anschlag) für die jeweilige Drehposition. Die Register für die Feuerknöpfe der Drehregler haben den Wert 0, wenn der Knopf gedrückt ist, und 1, wenn er es nicht ist.

Zur Abfrage der Maltafel muss man jeweils ein Paar von Drehreglern abfragen. Die beiden Reglerwerte geben dann die Position, die beiden Knopfwerte den Status der Kontrollknöpfe auf der Maltafel wieder. Der Kontrollknopf am Stift der Maltafel entspricht in der Abfrage der Steuerknüppelrichtung "oben".

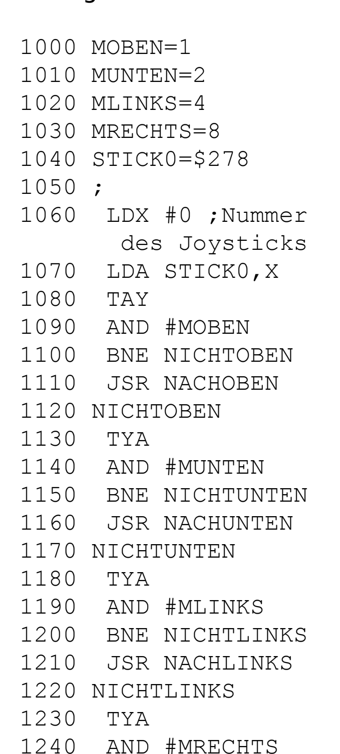
Bei den Feuerknöpfen der Joysticks ist es genauso wie bei denen der Drehregler: Eine 0 bedeutet "Knopf gedrückt", eine 1 bedeutet "Knopf nicht gedrückt".

Bei den Richtungsregistern ergibt sich das Problem, dass jeder Richtung ein Bit zugeordnet ist, sodass man bei diagonalen Richtungen zu, gelinde gesagt, unübersichtlichen Werten kommt. Hier zunächst die Belegung der Bits:

Bit	**Richtung**
0	oben
1	unten
2	links
3	rechts

Wenn der Steuerknüppel in die jeweilige Richtung gedrückt wird, nimmt das betreffende Bit den Wert 0 an, sodass man in Ruhestellung den Wert 15 vorfindet.

Hier eine kurze Beispielabfrageroutine, die ein Richtungsregister korrekt abfragt.



```
1000 MOBEN=1
1010 MUNTEN=2
1020 MLINKS=4
1030 MRECHTS=8
1040 STICK0=$278
1050 ;
1060  LDX #0 ;Nummer
       des Joysticks
1070  LDA STICK0,X
1080  TAY
1090  AND #MOBEN
1100  BNE NICHTOBEN
1110  JSR NACHOBEN
1120 NICHTOBEN
1130  TYA
1140  AND #MUNTEN
1150  BNE NICHTUNTEN
1160  JSR NACHUNTEN
1170 NICHTUNTEN
1180  TYA
1190  AND #MLINKS
1200  BNE NICHTLINKS
1210  JSR NACHLINKS
1220 NICHTLINKS
1230  TYA
1240  AND #MRECHTS
```

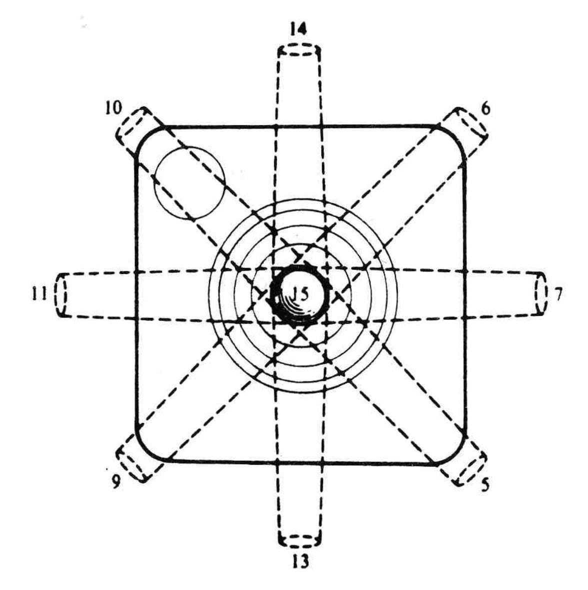


Abb. 2.22: Joystick-Positionen

```
1250  BNE NICHTRECHTS
1260  JSR NACHRECHTS
1270 NICHTRECHTS
1280 ;
```

2.18 Keyboard

Obwohl die Tastaturen der verschiedenen Modelle sehr unterschiedlich aussehen, ist die Haupttastatur stets funktionsgleich. Unterschiede entstehen durch Zusatztasten - die Funktions-Tasten beim 1200XL oder die Konsoltasten beim XL/XE. Unterschiede gibt es auch in der Hardware der Tastaturen. Die ATARI-Modelle 400, 1200XL, alle XE und das XEGS haben eine Tastaturfolie, die bei Defekt nur schwer zu ersetzen ist. Der 600XL besitzt als einziges Modell eine echte Schaltertastatur, die teilweise auch im 800XL verbaut wurde. Beim 800XL findet man daher beides, Schalter und Folien.

2.18.1 Allgemeiner Überblick

Die Tastatur des ATARI-Computers besteht aus zwei verschiedenen, räumlich getrennten Einheiten:

– der alphanumerischen Tastatur, zu der die Buchstabentasten, Ziffertasten etc. gehören,

– den Funktionstasten (HELP, START, SELECT, OPTION, RESET).

Diese Unterscheidung gibt es allerdings nur äußerlich. Vom internen Aufbau und daher auch für die Programmierung gilt folgende Aufteilung:

1. Die RESET-Taste kann zwar nicht im eigentlichen Sinne abgefragt werden, die Reaktion des Computers auf das Drücken der RESET-Taste lässt sich gleichwohl ändern.

2. START, SELECT und OPTION: die drei Funktionstasten, auch "Console"-Tasten genannt, sind von der restlichen Tastatur völlig unabhängig (→ CONSOL).

3. Auch die BREAK-Taste arbeitet völlig unabhängig von allen anderen Tasten (→ BREAK-Taste).

4. Die Tasten SHIFT und CONTROL erzeugen weder einen Interrupt noch einen eigenen Code, sondern beeinflussen lediglich die von den unter 5. aufgeführten Tasten erzeugten Werte.

5. Dies sind alle Tasten, die nicht unter 1. bis 4. aufgeführt sind. Auch die HELP-Taste und die zusätzlichen Funktionstasten des 1200XL gehören dazu!

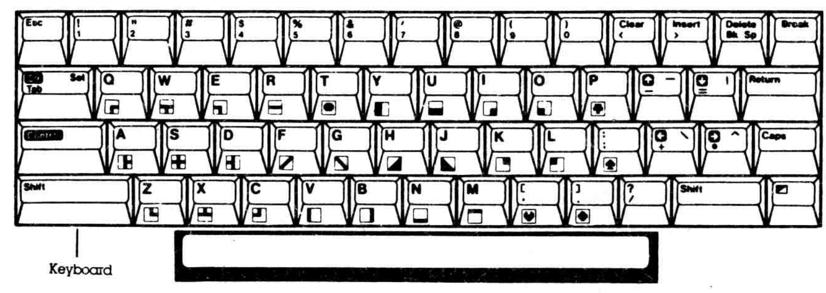


Abb. 2.23: Normaltastatur

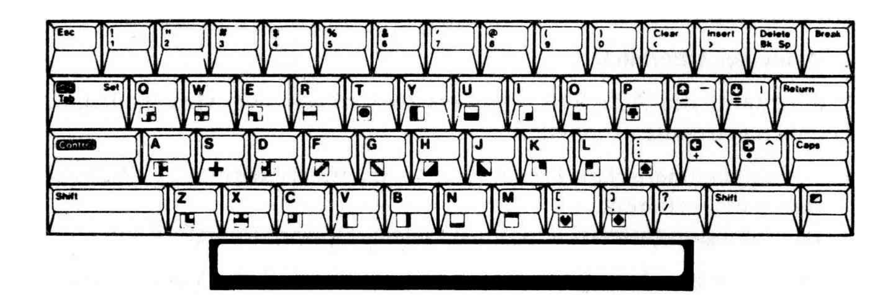


Abb. 2.24: Grafiktastatur

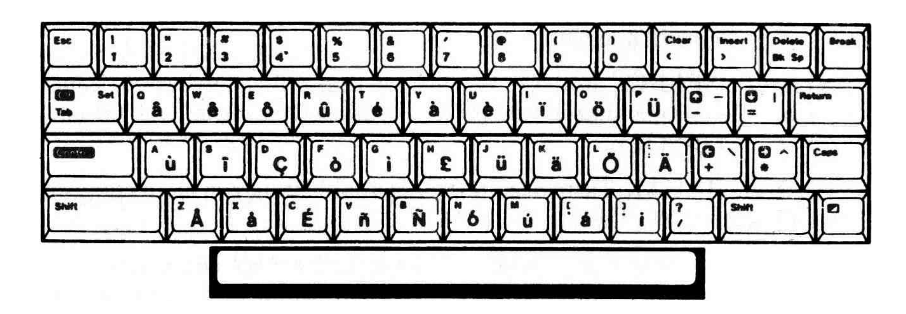


Abb. 2.25: Tastatur mit Umlauten

Im Folgenden soll nur die Funktionsweise der unter 4. und 5. aufgeführten Tasten beschrieben werden. Die Abfrage der anderen Teile der Tastatur wird jeweils in eigenen Abschnitten erklärt (RESET, BREAK-Taste, CONSOLE-Tasten).

Die Tasten SHIFT und CONTROL erzeugen keine eigenen Tastaturwerte, sondern verändern lediglich den Wert, der durch den Druck einer anderen Taste erzeugt wird. Während die CONTROL-Taste nicht einzeln abgefragt werden kann, ist dies mit den beiden Shift-Tasten, die sich untereinander in der Abfrage nicht unterscheiden, möglich. Siehe hierzu das Hardwareregister SKSTAT (53775; $D20F).

Das Drücken einer der betreffenden Tasten löst einen maskierbaren Interrupt aus, der über die Register POKMSK (16; $10) und IRQEN (53774; $D20E) ein- und ausgeschaltet werden kann. Als Folge dieses Interrupts wird die interne Keyboard-Interrupt-Routine aufgerufen, deren Adresse in VKEYBD (520,521; $208,$209) abgelegt ist. Dieser Vektor zeigt normalerweise auf die standardmäßige Routine im Betriebssystem-ROM, kann aber selbstverständlich auch auf eine eigene Routine gesetzt werden.

Daneben enthält das Register KBCODE (53769; $D209) den sogenannten Tastaturwert, der sich aus der Nummer der gedrückten Taste und der Information, ob SHIFT oder/und CONTROL gleichzeitig gedrückt worden sind, zusammensetzt. Dabei gibt das höchstwertigste Bit, also Bit 7, an, ob CONTROL gedrückt wurde und das nächste Bit, also Bit 6, ob gleichzeitig SHIFT gedrückt wurde. Die restlichen sechs Bits, die also noch 64 verschiedene Werte annehmen können, geben an, welche der Tasten gedrückt wurde.

Eine Liste aller erzeugbaren Tastaturwerte und der dazu notwendigen Tastenkombinationen findet sich in der großen Gesamttabelle im Nachschlagteil. Man beachte, dass es aufgrund der Tastenzahl (53 bei alten Geräten, 58 beim 1200XL und 54 bei allen übrigen Geräten) einige freie Tastaturwerte gibt, und dass nicht jede Taste in Verbindung mit SHIFT und CONTROL gleichzeitig einen Wert erzeugt.

Für das Verständnis der weiteren Bearbeitung von Tastendrücken ist die Kenntnis der Funktionsweise der normalen Tastatur-Interrupt-Routine notwendig.

2.18.2 Der Tastatur-Interrupt

Zunächst wird der Inhalt von KBCODE mit dem Wert des Registers CH1 (754; $2F2) verglichen, das jeweils den Tastaturwert des letzten Tastendrucks enthält. Wenn sich der Tastaturwert nicht verändert hat, wird

überprüft, ob der Zähler KEYDEL (753; $2F1), der während jedes Vertikal-Blank-Interrupts dekrementiert wird, den Wert 0 erreicht hat. Ist dies nicht der Fall, wird der Tastendruck als Tastenprellen angesehen und ignoriert. Es folgt die Überprüfung, ob es sich bei dem Tastendruck um die Tastenkombination CONTROL-F1 handelt, die sich allerdings nur auf der Tastatur eines 1200XL erzeugen lässt. In einem solchen Fall wird das Flag KEYDIS (621; $26D) ein- bzw. ausgeschaltet (und mithin die Tastatur ein- oder ausgeschaltet). Daraufhin wird nämlich eben dieses Flag überprüft, und in dem Fall, dass die Tastatur abgeschaltet ist, an das Ende der Routine verzweigt. Die Nutzung dieses Flags ist also auf allen XL/XE-Geräten dennoch möglich. Handelt es sich bei der gedrückten Tastenkombination um CONTROL-1, wird das Flag SSFLAG (767; $2FF) umgeschaltet. Dieses Flag wird vor sämtlichen Ausgaberoutinen für den Bildschirm abgefragt und in dem Fall, dass es den Wert 255 enthält, in eine Warteschleife verzweigt. Daher ist es möglich, Bildschirmausgaben über CONTROL-1 zu stoppen und fortzusetzen. Gehört die HELP-Taste zu den gedrückten Tasten, dann wird der Wert von KBCODE nun in HELPFG (732; $2DC) übertragen.

Es folgen nun noch Abfragen für die Funktionstasten des 1200XL, die es erlauben, über CONTROL-F2 die Bilderzeugung abzuschalten (was bei manchen Programmen erhebliche Geschwindigkeitsvorteile bringen kann; entspricht POKE 559,0) und über CONTROL-F4 zwischen dem normalen und dem internationalen Zeichensatz hin- und herzuschalten. Schließlich wird der Wert von KBCODE in CH1 (754; $2F2) und in CH (764; $2FC) übertragen, die Register für die Tastenentprellung und die Tastenwiederholfunktion initialisiert und die Inhalte der benutzten Register wiederhergestellt. Die Tasten F1-F4 sind im XL/XE-OS vorhanden und lassen sich benutzen, wenn man sie nachrüstet.

Schließlich ist noch ein Blick in die Vertikal-Blank-Routine vonnöten, die u.a. für die Tastenwiederholfunktion zuständig ist.

2.18.3 Tastenwiederholung durch den VBI

Zunächst wird über SKSTAT überprüft, ob überhaupt irgendeine Taste gedrückt ist. Ist dies nicht der Fall, wird der Zähler KEYDEL, sofern er noch nicht den Wert 0 erreicht hat, dekrementiert. Wenn nun tatsächlich noch die gleiche Taste gedrückt ist, kein Tastenprellen vorliegt, die Tastatur nicht abgeschaltet ist und auch die notwendige Frist seit der letzten Tastenwiederholung verstrichen ist, wird diese dadurch simuliert, dass erneut der Wert von KBCODE in CH übertragen wird. Von dieser Tastenwiederholung ausgeschlossen sind allerdings CONTROL-1, CONTROL-F1, CONTROL-F2, CONTROL-F4 und die HELP-Taste.

Der Ablauf der internen Tastatur-Interrupt-Routine und der Vertikal-Blank-Interrupt-Routine ist bei den alten Betriebssystemversionen (ATARI 400 und 800) insofern anders, als zusätzliche Fähigkeiten der XL/XE-Rechner wie Funktionstasten, die HELP-Taste, die Abschaltfunktion der Tastatur und die regelbare Tastenwiederholgeschwindigkeit nicht berücksichtigt werden.

2.18.4 Der Tastatur-Treiber

Im Betriebssystem des ATARI ist ein Gerätetreiber (K:) zur Abfrage der Tastatur integriert. Dieser Handler (→ Gerätetreiber) dient nur dem Einlesen von ATASCII-Werten von der Tastatur. Er wird betriebssystemintern auch vom Editor (E:) benutzt. Daher unterstützt der Treiber für die Tastatur auch nur die folgenden Operationen:

OPEN
CLOSE
GETBYTE
GETSTATUS

Nachfolgend eine Beschreibung der einzelnen Funktionen:

OPEN
Bis auf alle Vorgänge, die mit dem logischen Öffnen eines Kanals zu tun haben (zum Beispiel Belegtmeldung des verwendeten IOCB), hat dieses Kommando keine weiteren Auswirkungen.

CLOSE
Bis auf alle Vorgänge, die mit dem logischen Schließen eines Kanals zu tun haben (zum Beispiel Freimeldung des verwendeten IOCB), hat dieses Kommando keine weiteren Auswirkungen.

GETSTATUS
Diese Funktion liefert als Ergebnis stets den Wert 1 (d.h. "Status in Ordnung"; kein Fehler aufgetreten).

GETBYTE
Diese Funktion dient zum Einlesen einzelner ATASCII-Werte von der Tastatur. Als Tastaturregister wird dabei CH (764; $2FC) benutzt, welches ausschließlich von der Tastatur-Interrupt-Routine und dem Vertikal-Blank-Interrupt verändert wird. Dadurch sind automatisch alle Funktionen dieser Betriebssystemroutinen, wie zum Beispiel die Tastenwiederholfunktion, im Tastaturtreiber integriert. Eine weitere Folge davon ist, dass man durch Modifizieren der Tastatur-Interrupt-Routine auch die Arbeitsweise des Tastaturtreibers ändern kann.

Nun zum eigentlichen Ablauf der Routine GETBYTE:

Zunächst wird überprüft, ob die BREAK-Taste gedrückt ist. Ist dies der Fall, wird als ATASCII-Code der Wert 155 (Return) zurückgegeben und als Status 128 (Break-Taste gedrückt) erzeugt.

Daraufhin wird CH mit dem Wert 255 verglichen, um festzustellen, ob überhaupt eine Taste gedrückt worden ist. Ist noch kein Tastendruck erfolgt, wird wieder zum Anfang der Routine zurückgesprungen und darauf gewartet, dass CH durch die Tastatur-Interrupt-Routine einen neuen Wert erhält.

Enthält CH dann einen gültigen Tastaturcode (siehe oben) dann wird zunächst, falls durch NOCLIK (731; $2DB) nicht ausgeschaltet, ein Klickgeräusch erzeugt (gilt nicht für ATARI 400/800). Ist der Tastencode gleich oder größer als 192 ($C0), d.h. handelt es sich um einen Tastendruck mit SHIFT und CONTROL, wird wiederum an den Anfang der Routine verzweigt und somit dieser Tastendruck ignoriert.

Zur Umwandlung von Tastaturcodes in ATASCII-Werte (siehe auch Gesamttabelle im Tabellenteil) verfügt das Betriebssystem bei den XL/XE-Geräten über den Vektor KEYDEF (121,122; $7A,$7B), der auf den Anfang einer Konversionstabelle zeigt, die 192 Zeichen lang ist. In dieser Tabelle ist jeweils zu den Tastencodes zwischen 0 und 191 der entsprechende ATASCII-Wert abgespeichert. Da es einen Zeiger für diese Tabelle gibt, kann man auf sehr einfache Weise die Tastenbelegung ändern. Auch die "alten" Geräte verfügen über eine solche Tabelle, nicht aber über einen Vektor dafür, sodass eine Modifizierung der Tastenbelegung nicht ohne Weiteres möglich ist.

Bei den Werten in dieser Tabelle handelt es sich nicht nur um ATASCII-Codes (von denen ja nur der Bereich von 0 bis 127 benötigt wird, weil die mögliche Invertierung ja nicht vom letzten Tastendruck abhängt und erst später anhand von INVFLG (694; $2B6) vorgenommen wird), sondern auch um spezielle Steuerzeichen für den Tastaturtreiber. Diese Codes liegen im Bereich von 128 ($80) bis 145 ($91) (→ Tab. 2.18).

Die Codes 138-145 entsprechen genaugenommen den Funktionstasten F1-F4 und Shift-F1-F4. Beim Drücken dieser Tasten werden die Codes der 8 Byte großen Tabelle ausgeführt, auf die FKDEF (96,97; $60,$61) zeigt. Das sind normalerweise 28-31 (Cursortasten) und 142-145 (Cursorrandpositionierung).

Liegt der aus dieser Tabelle gelesene Wert unter 128 ($80), dann handelt es sich also um einen "normalen" Tastendruck. Daher wird nur noch die

Veränderung des ATASCII-Wertes anhand der Register INVFLG (Flag für Inversdarstellung) und SHFLOK (Flag für Groß-/Kleinschrifteinstellung) vorgenommen und die GETBYTE-Routine verlassen.

Code		
Dez	Hex	Wirkung
128	$80	ungültige Tastenkombination
129	$81	Umschalten zwischen Invers- und Normalschrift
130	$82	Umschalten zwischen Groß- und Kleinschrift
131	$83	Einschalten des Großschriftmodus
132	$84	Einschalten des Grafikzeichenmodus
133	$85	End-of-File (Dateiende; Drücken von CTRL-3)
134	$86	unbenutzt
135	$87	unbenutzt
136	$88	unbenutzt
137	$89	Tastaturklick ein-/ausschalten
138	$8A	Cursor eine Zeile nach oben bewegen
139	$8B	Cursor eine Zeile nach unten bewegen
140	$8C	Cursor ein Zeichen nach links bewegen
141	$8D	Cursor ein Zeichen nach rechts bewegen
142	$8E	Cursor in die linke obere Bildschirmecke ("home")
143	$8F	Cursor in die linke untere Bildschirmecke
144	$90	Cursor auf den linken Rand
145	$91	Cursor auf den rechten Rand

Tab. 2.18: Codes für die zusätzlichen Editierfunktionen

Ansonsten ist dieser Wert der ATASCII-Code einer Cursor-Steuerfunktion, der über 128 liegt, oder eben einer der speziellen Codes für den Editor. Über diese Sondercodes wird die Umschaltung mit der Caps-Taste vorgenommen, der Tastaturklick ein- und ausgeschaltet, die Inversdarstellung ein- und ausgeschaltet oder die Meldung eines Datei-Ende-Fehlers, den man durch Drücken von CONTROL-3 erzeugen kann, vorgenommen. Je nach Bedeutung dieser Codes für Sonderfunktionen wird daraufhin eine

bestimmte Funktion, wie etwa das Umschalten des Flags für Normal- und Inversdarstellung, ausgeführt oder das Drücken einer bestimmten anderen Taste "simuliert". Beispiele hierfür sind die Bearbeitung von CONTROL-3, das ein Return-Zeichen (ATASCII-Code 155; $9B) generiert, oder die vier zusätzlichen Cursorbewegungsfunktionen, die sich allerdings nur beim 1200XL über die Tastatur erzeugen lassen: Hierbei wird jeweils der ATASCII-Code der "normalen" Cursorfunktion (28-31; $1C-$1F), die den Cursor um nur ein Zeichen in die gewünschte Richtung bewegt, erzeugt. Als Zeichen für den Editor, dass es sich in Wahrheit um eine andere Funktion handelt, wird zusätzlich das Flag SUPERF (1000; $3E8) auf 1 gesetzt.

2.19 Kompatibilität

Bis zum Produktionsende in 1992 erschienen auf dem deutschen Markt acht verschiedene ATARI-Modelle. Sie unterscheiden sich in Ausstattung und durch das eingebaute Betriebssystem. Die XL/XE-Modelle und das XE Game System (XEGS) sind weitestgehend funktionsgleich. Die ATARI 400/800 spielen heute keine Rolle mehr bei der Programmentwicklung.

Generell gilt, dass Software, die spezielle Features wie z.B. die zusätzlichen 64 KByte des 130XE oder die 256 KByte große Speichererweiterung eines aufgerüsteten ATARI 800 nutzt, so geschrieben werden muss, dass das Vorhandensein der benötigten Hardware überprüft wird.

Beginnen wir mit den Unterschieden in der Hardware:

	400	800	1200XL	600XL	800XL	alle XE	XEGS
Slots	—	4		—	—	—	—
Modulports	1	2	1	1	1	1	1
Monitorausgang	—	X	X	X	X	X	X
Controller-Ports	4	4	2	2	2	2	2
Systembus	—	—	—	PBI	PBI	ECI	—
Funktionstasten	—	—	X	—	—	—	—
Help-Taste	—	—	X	X	X	X	X
Internes BASIC	—	—	—	X	X	X	X
externe Tastatur	—	—	—	—	—	—	X
Lautsprecher	X	X			(wird simuliert)		

- Kapitel 4 enthält die unterschiedlichen Belegungen von PBI und ECI.

- Die Funktionstasten können zusätzlich eingebaut werden, da sie im Betriebssystem verankert sind.

Zu den Punkten im Einzelnen:

Slots - d.h. Steckplätze für Erweiterungskarten - gab es nur beim ATARI 800. Alle Erweiterungen hierfür - beispielsweise eine 80-Zeichen-Karte, eine Echtzeituhr oder eine 128-KByte-RAM-Erweiterung - lassen sich also nur auf dem ATARI 800 verwenden.

Modulanschlüsse: Der ATARI 800 verfügt über einen zusätzlichen Modulschacht (rechts), der praktisch kaum verwendet worden ist ("rechte" Module lassen sich nicht im "linken" Modulschacht bzw. dem Cartridgeslot der anderen ATARI-Computer verwenden). Beispielsweise wurde eine BASIC-Erweiterung angeboten, die mit dem ATARI-BASIC-Modul zusammenarbeitet (war auch für XL/XE-Modelle erhältlich).

Controller-Ports: Der ATARI 400 und 800 verfügten über insgesamt vier Controller-Ports (Joystickanschlüsse), sodass es möglich war, zu viert gleichzeitig ein Programm zu steuern (man denke an MULE oder BASKETBALL). Das Problem, dass auf den neueren Geräten nur noch zwei Joysticks angeschlossen werden können, wurde folgendermaßen gelöst:

Das Betriebssystem schreibt die Werte für die Joysticks 1 und 2 sowohl in die Register von Joystick 1/2, als auch in die von 3/4. Das heißt, dass man für Joystick 3 und 4 stets die Werte der ersten beiden Joysticks erhält. Dies nützt allerdings nur dann, wenn die vier Steuerknüppel nicht gleichzeitig verwendet werden müssen und das fragliche Programm die vom Betriebssystem unterstützten Schattenregister verwendet.

Probleme können auch dann auftreten, wenn versucht wird, über Joystickanschluss 3 und 4 Daten auszugeben. In diesem Fall wird in den XL/XE-Geräten die interne Speicheraufteilung verändert, was zu sofortigen Problemen führt (Beispiel: das Betriebssystem wird ausgeschaltet, ohne dass im dahinter liegenden RAM ein Ersatz vorliegt).

Systembus: Beim 600XL und 800XL ist der gesamte Systembus (→ Parallel Bus Interface - PBI) durch eine Buchse an der Gehäuserückseite zugänglich. Bei den XE-Modellen gibt es stattdessen die Kombination aus Modulschacht und ECI (Enhanced Cartridge Interface), die gemeinsam den Systembus bilden. Die unterschiedliche Belegung der Steckanschlüsse ist in Kapitel 4 (→ Buchsenbelegungen) erläutert. Für den PBI gibt es von ATARI nur das 1064 (64-KByte-Speichererweiterung für den 600XL). Geplant waren außerdem eine 80-Zeichen-Karte und ein paralleles Interface für Diskettenlaufwerke. Diese und andere Erweiterungen für den PBI (ATARI 1090) wurden nicht mehr auf den Markt gebracht. Für den ECI wurde von ATARI niemals ein Gerät konzipiert. Allerdings gab es eine ganze Reihe an Geräten von anderen Anbietern. Diese sind heute meist

nur noch gebraucht zu bekommen. Ausnahmen davon sind die Festplatten-Interfaces MIO, das neu aufgelegt wurde, und das neu entwickelte KMK/JZ IDE V 2.0 Plus (2011).

HELP-Taste: Die HELP-Taste gibt es nur auf den Geräten der XL- und XE-Serie. Sie erzeugt einen Tastaturwert, der sich auf dem ATARI 400 und 800 durch keine irgendwie geartete Tastenkombination erzeugen lässt. Die meisten Programme, die die HELP-Taste benutzen, bieten allerdings für die älteren Geräte eine alternative Tastenkombination an.

Ein weiterer Grund für Inkompatibilität liegt in den unterschiedlichen Betriebssystemen, die in die Rechner eingebaut sind. Werden bei der Programmierung einige wenige Regeln beachtet, die übrigens von Anfang an bekannt waren, so läuft das Programm unter allen existierenden und künftigen Betriebssystemen.

1. Regel: Speicherstellen, die im Speicherplan als "unbenutzt" gekennzeichnet sind, dürfen auf keinen Fall verwendet werden – schon bei der nächsten Betriebssystemversion könnten sie benutzt sein. Es sollten nur Speicherstellen und -bereiche benutzt werden, die ausdrücklich als "frei" gekennzeichnet sind (auch einzelne "freie" Bits können in neuen Geräten eine Bedeutung erlangen - daher: Vorsicht bei Bit-Manipulationen an Systemadressen).

2. Regel: Betriebssystem-Routinen nur über die Einsprungvektortabellen aufrufen – alles andere kann sich verändern. Die Benutzung der Vektortabellen für die residenten Treiber (Editor, Bildschirm, Kassette, Drucker) ist zwar grundsätzlich erlaubt, kann aber zu Problemen führen, wenn das fragliche Peripheriegerät nicht richtig initialisiert ist!

Programme, die allein deshalb laufen, weil sie "illegale" Einsprünge in das Betriebssystem vornehmen, kann man dennoch "reparieren": Dazu gibt es von ATARI die "Translator"-Diskette, die in den Speicherbereich von 57344-65535 ($E000-$FFFF) (es sind also mindestens 64KByte-RAM nötig) das Betriebssystem der 400/800-Reihe lädt.

2.20 Page 0

Page 0 (Seite 0) des Speichers ist von besonderer Bedeutung, da man nur Page-0-Register für die indirekte Adressierung des 6502-Prozessors verwenden kann. Darüber hinaus kann man oft durch Verwendung von Variablen auf Page 0 eine Menge Speicherplatz (im Programm) sparen.

Aufgrund der Wichtigkeit von Page 0 ist es für professionelle Programmentwicklungen sehr nützlich, wenn einerseits der benutzte und der unbe-

nutzte Teil von Page 0 klar voneinander abgetrennt sind und andererseits eben ein möglichst großer Teil von Page 0 unbenutzt ist.

Dies ist beim ATARI weitgehend geschehen. Die erste Hälfte von Page 0, also die Bytes 0-127 ($0-$7F), ist für das Betriebssystem reserviert. Die andere Hälfte ist für Anwenderprogramme frei. Unter Anwenderprogrammen sind allerdings nicht nur eigene Programme zu verstehen, sondern eben alles, was nicht direkt zum Betriebssystem gehört - dazu gehören unter anderem auch ATARI-BASIC und das Fließkommapaket. Unter ATARI-BASIC sind die Bytes 203-209 ($CB-$D1) frei und werden oft genutzt. Als für Maschinensprache-Unterprogramme frei kann man allerdings alle Fließkommavariablen ansehen, soweit der USR-Aufruf nicht innerhalb eines mathematischen Ausdrucks steht und das Programm nicht selbst Fließkommaroutinen benutzt (→ Fließkomma-Arithmetik).

2.21 Page 6

Page 6 (Seite 6) des Speichers reicht von 1536-1791 ($600-$6FF) und wird weder vom Betriebssystem noch vom Diskettenbetriebssystem benutzt. So kann man Page 6 zum Beispiel unter BASIC für eigene Unterprogramme oder Daten verwenden. Man sollte allerdings beachten, dass gerade deshalb diese Stelle des Speichers sehr gerne benutzt wird, sodass man genau darauf achten sollte, dass nicht schon ein anderer Programmteil Page 6 verwendet. Außerdem hängt es stets vom Anwenderprogramm ab, ob Page 6 tatsächlich frei ist - unter MAC/65 (Warenzeichen von Optimized Systems Software) ist beispielsweise nur die zweite Hälfte für eigene Zwecke frei!

2.22 Player-Missile-Grafik

In Ergänzung der Grafikstufen, die durch den ANTIC (→ ANTIC) erzeugt werden können, gibt es noch zusätzlich die sogenannte Player-Missile-Grafik (PM-Grafik). Bei der Erzeugung der PM-Grafik kooperieren der ANTIC, der die darzustellenden Bilddaten mittels direktem Speicherzugriff (DMA) aus dem Speicher liest, und der GTIA (Grafik Television Interface Adapter), der diese Bilddaten auf dem Bildschirm darstellt.

Die Vorteile von Objekten, die völlig unabhängig vom normalen Bild über den Bildschirm gelegt werden können, liegen auf der Hand:

- bis zu sechs zusätzliche Farben können ohne Interrupts auf den Bildschirm gebracht werden;
- für die Bewegung dieser Objekte braucht der Hintergrund nicht verändert zu werden;
- es wird eine automatische Kollisionsüberprüfung vorgenommen.

Wie funktioniert die Player-Missile-Grafik genau?
Insgesamt gibt es vier Player mit einer Breite von jeweils acht Punkten und vier Missiles mit einer Breite von jeweils zwei Punkten. Jedes Player-Missile-Paar hat eine eigene Farbe. Jedes der Objekte ist, je nach Bildschirmauflösung, 128 oder 256 Zeilen hoch und erstreckt sich damit über die gesamte Höhe des erzeugten Bildes. Für jeden Player ist somit ein 128 bzw. 256 Bytes langer Speicherblock reserviert, wobei das erste Byte in der obersten Zeile des Players abgebildet wird. Vertikale Bewegungen müssen somit durch Verschiebungen der Daten in diesem Speicherblock verwirklicht werden. Die Codierung des Punktmusters erfolgt genau wie bei der Definition eines Zeichensatzes: Das höchstwertigste Bit kontrolliert den Punkt am linken Rand usw. (siehe Abb. 2.26).

Für die Player-Missile-Grafik braucht man somit, je nach benutzter Auflösung, 640 bzw. 1280 Bytes RAM. Jedes der insgesamt acht Objekte kann in einfacher (entspricht zwei Zeichen in GRAPHICS 0), zweifacher und vierfacher Breite dargestellt werden. Dabei muss allerdings beachtet werden, dass dabei nicht etwa die horizontale Auflösung vergrößert wird, sondern vielmehr die Breite der einzelnen Bildpunkte größer wird (siehe Diagramm "Definition von Players"). Jedes der Objekte kann eine von 256 horizontalen Positionen einnehmen, die allerdings nicht alle auf dem Bildschirm zu erkennen sind, da sie auch horizontal den sichtbaren Bereich des Bildschirms verlassen können (siehe Abb. 2.29). Daneben unterstützt der GTIA auch die Erkennung von Kollisionen für praktisch jede Kombination zwischen Players, Missiles und Hintergrundfarben.

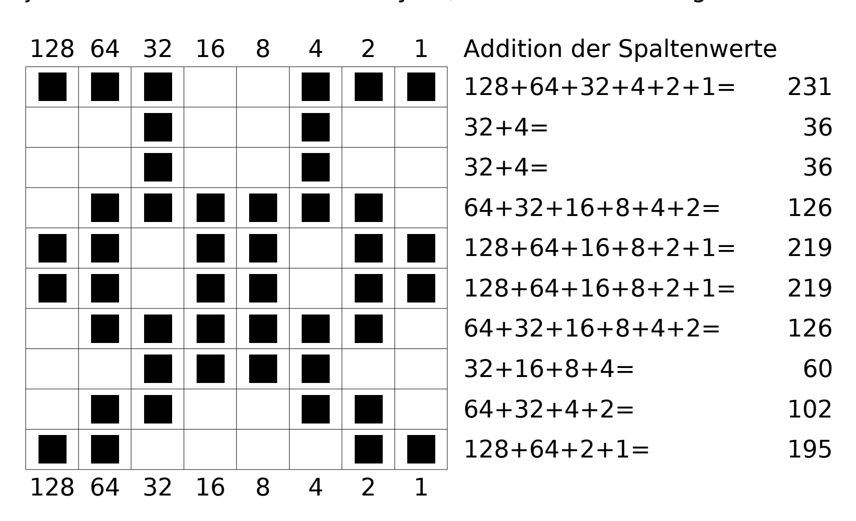


Abb. 2.26: Entwurfsformular für Player

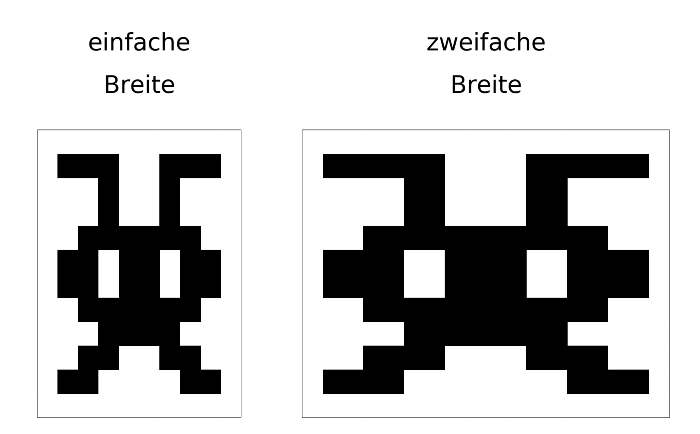


Abb. 2.27: Definition von Players

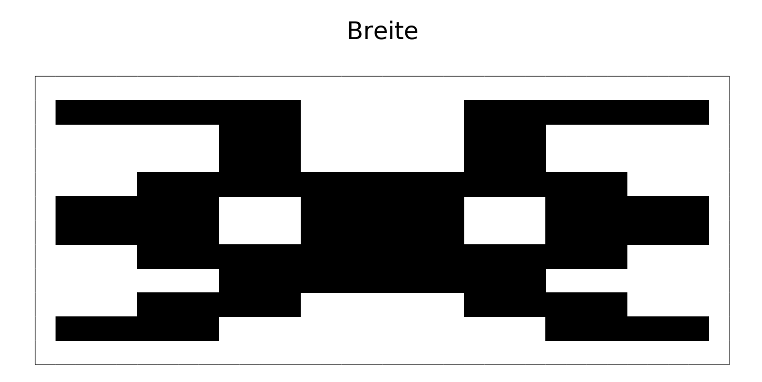


Abb. 2.28: Definition von Players (Forts.)

Im Folgenden zunächst eine Übersicht über alle für die Player-Missile-Grafik wichtigen Register:

SDMCTL $22F 559
DMACTL $D400 54272

Über das Hardwareregister DMACTL (und das dazugehörige Schattenregister SDMCTL) werden die verschiedenen DMA-Modi und die Auflösung (einzeilige oder zweizeilige) ausgewählt. Die Belegung der für die PM-Grafik wichtigen Bits ist wie folgt:

Bit 2: Missilegrafik ein
Bit 3: Playergrafik einschalten (schaltet auch Missilegrafik an)
Bit 4: 0=zweizeilige, 1=einzeilige Auflösung einschalten

In der zweizeiligen Auflösung ist jeder Bildpunkt in einem Player oder Missile doppelt so hoch wie eine GRAPHICS-8-Zeile. Jeder Player belegt dann 128 Bytes und der gesamte Speicherplatzbedarf sinkt auf (siehe Diagramm "Aufteilung des Player-Missile-Arbeitsspeichers") die Hälfte.

GPRIOR	**$26F**	**623**
PRIOR	**$D01B**	**53275**

Die untersten vier Bits von GPRIOR und dem dazugehörigen Hardwareregister PRIOR kontrollieren, welches Objekt bzw. welche Anzeigefeld-Farbe bei Überlappungen zuoberst abgebildet werden soll. Dabei sollten keine Kombinationen der vier Alternativen verwendet werden, da sonst in den Überlappungsbereichen die Farbe Schwarz erzeugt wird.

Hier die Bedeutungen der vier möglichen Einstellungen:

Bit 0 gesetzt: Player 0-3, Anzeigefeld 0-3, Hintergrund
Bit 1 gesetzt: Player 0+1, Anzeigefeld 0-3, Player 2+3, Hintergrund
Bit 2 gesetzt: Anzeigefeld 0-3, Players 0-3, Hintergrund
Bit 3 gesetzt: Anzeigefeld 0+1, Player 0-3, Anzeigefeld 2+3, Hintergrund

Durch Setzen von Bit 4 können die vier Missiles zu einem fünften Player zusammengeschaltet werden. Einzige Folge davon ist, dass die Missiles nicht mehr die Farbe des dazugehörigen Players, sondern diejenige aus COLOR3 (711; $2C7), annehmen. Positionen und Breite müssen also nach wie vor einzeln eingestellt werden, was in Spezialfällen sogar die Flexibilität erhöhen kann.

Bit 5 dient zum Einschalten der mehrfarbigen Player-Missile-Grafik. In diesem Modus nehmen jeweils die Überlappungszonen von Player 0 und 1 bzw. Player 2 und 3 einen dritten Farbton an. Dieser Farbton ergibt sich aus den beiden Farben der beiden Players, die logisch oderiert werden.

PCOLR0-3	**$2C0-$2C3**	**704-707**
COLOM0-3	**$D012-$D015**	**53266-53269**

Dies sind die Farbregister für Players und Missiles 0 bis 3 (jedes Missile hat jeweils die gleiche Farbe wie der dazugehörige Player). Genau wie bei den anderen Farbregistern errechnet sich ein Farbwert wie folgt:

Farbwert=Farbnummer*16 +Helligkeit

HPOSP0-3 $D000-$D003 53248-53251
Register für die horizontalen Positionen der Players. Man beachte dabei, dass sich die Positionsangabe auf die linke Kante des Players bezieht (siehe Abb. 2.29).

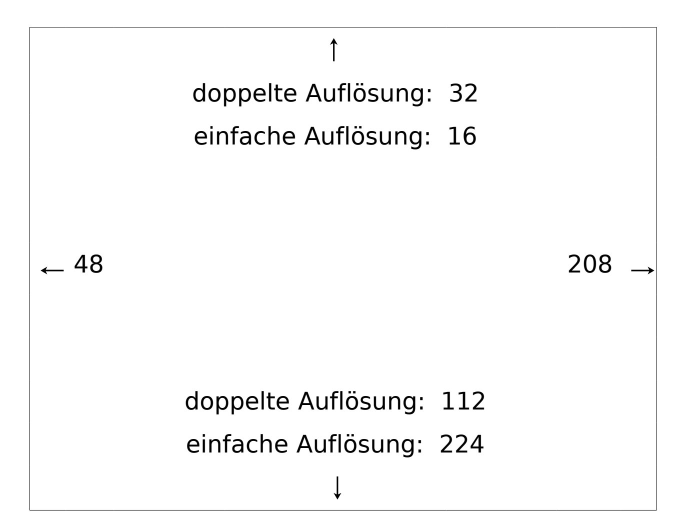


Abb. 2.29: Positionen der Player auf dem GRAPHICS x-Bildschirm

HPOSM0-3 $D004-$D007 53252-53255
Register für die horizontalen Positionen der Missiles.

SIZEP0-3 $D008-$D00B 53256-53259
Register für die Breiten der einzelnen Players. Mögliche Werte sind dabei 0 (normale Breite), 1 (doppelte Breite) und 3 (vierfache Breite). Siehe dazu auch Abb. 2.35 "Definition von Players".

SIZEM $D00C 53260
Auch für die vier Missiles lassen sich die Breiten separat verstellen. Allerdings gibt es zu diesem Zweck nur dieses eine Register, in dem je ein Bit-Paar für ein Missile zuständig ist. Die Zuordnung der Bits ist wie folgt:

Bit 7+6: Missile 3
Bit 4+5: Missile 2
Bit 2+3: Missile 1
Bit 0+1: Missile 0

Wiederum sind normale Breite (Bitmuster: 00), doppelte Breite (Bitmuster: 01) und vierfache Breite (Bitmuster: 11) möglich.

GRAFP0-3 $D00D-D010 53261-53264
Normalerweise hat der ANTIC die Aufgabe, über DMA aus dem Speicher die Daten für die Player-Missile-Grafik zu lesen und dann dem GTIA zur Verfügung zu stellen. Alle weiteren Aufgaben wie Farberzeugung, Kollisionsüberprüfung etc. werden vom GTIA übernommen. Man kann nun auch die Steuerung des ANTIC ausschalten (über SDMCTL (022F, 559)) und die Bereitstellung der Bilddaten selbst übernehmen.

Das Punktmuster, das im Player auf dem Bildschirm erscheinen soll, muss in dieses Register geschrieben werden. Der GTIA erzeugt dieses Muster so lange, bis ein anderer Wert hineingeschrieben wird. Wollte man also echte Objekte erzeugen, müsste man nach jeder Bildzeile einen neuen Wert hineinschreiben (was selbstverständlich einerseits ein riesiger Programmieraufwand und andererseits ein großes Timing-Problem wäre). Eine sinnvolle Anwendung dieser Möglichkeit wäre allerdings die Erzeugung von Bildrändern (immerhin kann man damit 1 bzw. 2 KByte RAM sparen). Auch die Darstellung einer expandierenden Explosionswolke (siehe Screenshots von NADRAL – Abb. 2.30 und 2.31) ist über eine kurze Routine, die GRAFP0 und HPOSP0 in sehr kurzen Zeitabständen verändert, sehr simpel.

GRAFM $D011 53265
Kontrollregister für das für die Missiles erzeugte Punktmuster (siehe GRAFP0).

VDELAY $D01C 53276
Einer der Nachteile der geringeren, zweizeiligen, Auflösung der Player-Missile-Grafik ist, dass vertikale Bewegungen natürlich nur in halb so feinen Abstufungen möglich sind. VDELAY ermöglicht es, jeden der Players und Missiles um genau eine Zeile tiefer zu setzen, sodass man über eine Kombination zwischen einer normalen Verschieberoutine und regelmäßigem Umschalten von VDELAY eine feinere vertikale Bewegung erreichen kann. Die Zuständigkeit der einzelnen Bits ist folgendermaßen:

Bit	Objekt
7	Player 3
6	Player 2
5	Player 1
4	Player 0
3	Missile 3
2	Missile 2
1	Missile 1
0	Missile 0



Abb. 2.30: Nadral



Abb. 2.31: Nadral – Explosionswolke

GRACTL $D01D 53277

Grafikkontrollregister für den GTIA, über den dieser angewiesen werden kann, die Player/Missile-Daten vom ANTIC zu übernehmen statt aus den GRAFxx-Registern. Die relevanten Bits:

Bit 0: Erzeugung der Missiles einschalten
Bit 1: Erzeugung der Players einschalten

HITCLR $D01E 53278

Durch Einschreiben eines beliebigen Wertes in dieses Register können die Kollisionsregister gelöscht werden. Wozu das? Der GTIA setzt die entsprechenden Bits in den Kollisionsregistern jedes Mal dann, wenn die dazugehörige Kollision stattgefunden hat. Da das Register jedoch nicht wieder gelöscht wird, ist es möglich, die Kollision auch noch dann festzustellen, wenn sich die verantwortlichen Objekte schon weiterbewegt haben. Der GTIA setzt die Kollisionsregister immer in dem Moment, in dem er bei der Bilderzeugung die Überlappung "bemerkt". Daher ist es ratsam, jeweils zwischen dem letzten Löschen der Kollisionsregister und der nächsten Abfrage auf Kollision mindestens 1/50 Sekunde, also die Zeit, die für den Aufbau eines Bildes benötigt wird, verstreichen zu lassen.

PMBASE $D407 54279

In diesem Register muss die Anfangsadresse des Player-Missile-Arbeitsspeichers, der bei einzeiliger Auflösung 2048 Bytes und bei zweizeiliger Auflösung 1024 Bytes umfasst (siehe Abb. 2.32 "Aufteilung des Player-Missile-Arbeitsspeichers"), abgelegt werden. Der Anfang des Player-Missile-Speichers muss bei der geringeren Auflösung auf einer 1KByte-Grenze liegen (also ein Vielfaches von 1024 sein), bei der einzeiligen Auflösung auf einer 2KByte-Grenze liegen (also ein Vielfaches von 2048 sein). Das niederwertige Byte ist daher stets 0 und so gibt es auch nur ein Adressregister für das höherwertige Byte.

M0PF-M3PF $D000-$D003 53248-53251

Im Gegensatz zu anderen Grafikchips hat der GTIA die Fähigkeit, nicht nur zu erkennen, dass eine Überlappung (Kollision) stattgefunden hat, sondern auch festzustellen, welche Objekte miteinander kollidiert sind. Diese vier Register dienen dazu, Kollisionen zwischen den Missiles und den Anzeigefeldern anzuzeigen, wobei M0PF für Missile 0, M1PF für Missile 1, M2PF für Missile 2 und M3PF für Missile 3 zuständig ist.

Zweizeilige Auflösung Einzeilige Auflösung

- Adressen jeweils als Offset von PMBASE -

Belegung der einzelnen Bits							
0	1	2	3	4	5	6	7

Zweizeilige Auflösung					Einzeilige Auflösung
+0 →	unbenutzt – kann beliebig anders verwendet werden				← +0
+384 →	Missile 0	Missile 1	Missile 2	Missile 3	← +768
+512 →	Player 1				← +1024
+640 →	Player 2				← +1280
+768 →	Player 3				← +1536
+896 →	Player 4				← +1792
+1024 →					← +2048

Abb. 2.32: Aufteilung des Player-Missile-Arbeitsspeichers

Da es nur vier verschiedene Anzeigefelder gibt, sind auch nur jeweils vier Bits dieser Kollisionsregister benutzt:

Bit	**Anzeigefeld**
4-7	unbenutzt
3	3
2	2
1	1
0	0

Die Bits werden dann gesetzt, wenn eine Überlappung stattgefunden hat, und gelöscht, wenn ein beliebiger Wert in HITCLR geschrieben wird.

P0PF-P3PF $D004-$D007 53252-53255
Kollisionsregister für Kollisionen zwischen Players und Anzeigefeld (siehe unter M0PF).

M0PL-M3PL $D008-$D00B 53256-53259
Kollisionsregister für Kollisionen zwischen Missiles und Players. Die Verwendung der einzelnen Bits ist folgendermaßen:

Bit	Player
4-7	unbenutzt
3	3
2	2
1	1
0	0

P0PL-P3PL $D00C-$D00F 53260-53263
Kollisionsregister für Kollisionen von Players untereinander (siehe unter M0PL).

Für die Player-Missile-Grafik gibt es neben der üblichen Darstellung kleiner, bewegter Objekte noch viele andere Anwendungsmöglichkeiten:

Da die vier Player über eigene Farbregister verfügen, kann man durch Verwendung von Player-Missile-Grafik die Anzahl der Farben auf dem Bildschirm nahezu verdoppeln. Weiterhin ist es möglich, Programme, die eigentlich nur die normale Textdarstellung benutzen, bunter und übersichtlicher zu gestalten. Ein Beispiel hierfür ist "Bundesliga" (AXIS Komputerkunst), bei dem mithilfe von Player-Missile-Grafik Umrahmungen und Ähnliches erzeugt werden (siehe Abb. 2.33 - 2.35).

Eine weitere interessante Eigenschaft der Player-Missile-Grafik ist, dass sie sich nicht nur über die gesamte Höhe des Bildschirms erstrecken kann, sondern dass man, wenn man alle Objekte auf vierfache Breite setzt, auch horizontal die Breite eines normalen Bildschirms abdecken kann. Daraus ergeben sich ganz andere Nutzungsmöglichkeiten wie zum Beispiel die gleichzeitige Darstellung einer beliebigen Grafikstufe zusammen mit einer Pixel-Grafik mit der zugegeben etwas seltsamen Auflösung von 40*256 Bildpunkten.

Eine weitere lohnende Anwendung ist die farbige Unterlegung von Textbildschirmen wie bei dem Programm MEMO-BOX (AXIS Komputerkunst). Auf diese Art und Weise kann man mithin die Anzahl der Farben in der normalen Textgrafik deutlich erhöhen (Abb. 2.36 – 2.38).

6. Spieltag	Datum: 29.09.84		
Hamburger SV	1.FC K'lautern	3	2
Eintr.Frankfurt	Arminia Bielef.	3	0
F. Duesseldorf	Werder Bremen	3	2
SVW Mannheim	Bayer Uerdingen	2	1
VFB Stuttgart	Bayern Muenchen	1	3
1.FC Koeln	B.Dortmund	6	1
VfL Bochum	Leverkusen	0	0
Moenchengladb.	Karlsruher SC	3	3
Schalke 04	Eintr.Br'schw.	3	2

vorwärts blättern

ESC → zurück • CTRL-D → ausdrucken

Abb. 2.33: Bundesliga

Fussballbundesliga 1984 / 1985				
1.	Bayern Muenchen	68: 36	+ 32	41:17
2.	Werder Bremen	76: 46	+ 30	39:19
3.	1.FC Koeln	57: 49	+ 8	35:23
4.	Moenchengladb.	65: 43	+ 22	34:24
5.	Hamburger SV	50: 40	+ 10	33:23
6.	SVW Mannheim	41: 40	+ 1	33:25
7.	Bayer Uerdingen	50: 42	+ 8	31:25
8.	VFB Stuttgart	70: 49	+ 21	30:28
9.	VfL Bochum	45: 43	+ 2	29:27
10.	Leverkusen	44: 43	+ 1	27:31
11.	Schalke 04	53: 57	- 4	27:31
12.	Eintr.Frankfurt	54: 59	- 5	27:31
13.	1.FC K'lautern	37: 47	- 10	26:30
14.	F. Duesseldorf	46: 59	- 13	24:34
15.	B.Dortmund	41: 56	- 15	24:34
16.	Arminia Bielef.	37: 55	- 18	22:36
17.	Karlsruher SC	41: 76	- 35	18:40
18.	Eintr.Br'schw.	33: 68	- 35	18:40

ESC → zurück • CTRL-D → ausdrucken

Abb. 2.34: Bundesliga

Werder Bremen

Heim		Auswärts
1: 0	Bayer Uerdingen	3: 1
4: 2	Bayern Muenchen	4: 2
6: 2	1.FC Koeln	3: 2
3: 1	VFB Stuttgart	1: 3
1: 1	SVW Mannheim	1: 1
2: 1	F. Duesseldorf	3: 2
3: 3	Eintr.Frankfurt	1: 3
2: 1	Schalke 04	2: 2
5: 2	Hamburger SV	2: 0
2: 0	Moenchengladb.	1: 1
2: 2	VfL Bochum	1: 3
2: 2	Leverkusen	0: 0
7: 1	Karlsruher SC	
	1.FC K'lautern	2: 2
4: 1	Eintr.Br'schw.	
	Arminia Bielef.	3: 4
6: 0	B.Dortmund	

Mögliche Eingaben: A-T, CTRL-D, ESCape

Abb. 2.35: Bundesliga



Abb. 2.36: MEMO-BOX-Bildschirm



Abb. 2.37: MEMO-BOX-Bildschirm

Abb. 2.38: MEMO-BOX-Bildschirm

Zu guter Letzt sei noch erwähnt, dass man selbstverständlich alle Hardwareregister des GTIA und des ANTIC auch im Displaylist-Interrupt (→ ANTIC) verändern kann, sodass man durch Umschaltung der horizontalen Positionen einen Player förmlich zerschneiden und so übereinander weit mehr als vier verschiedene Player darstellen kann.

2.23 Schattenregister

Schattenregister sind Variablen, die von der internen Vertikal-Blank-Interrupt-Routine benutzt werden. Dabei muss man zwischen zwei verschiedenen Typen unterscheiden:

Zunächst gibt es Schattenregister, deren Wert jede 1/50 Sekunde in das betreffende Hardwareregister (→ Hardwareregister) übertragen wird. Ein Beispiel dafür sind die Farbregister: Wenn man den Wert der Hardware-Farbregister selbst ändern wollte, würde es häufig zu Bildstörungen kommen. Dies sollte man normalerweise nur dann tun, wenn die Bildschirmerzeugung gerade unterbrochen ist. Das Betriebssystem sorgt nun dafür, dass die Hardware-Farbregister immer erst dann verändert werden, wenn gerade die Erzeugung eines Bildes abgeschlossen ist. Außerdem hat dies den Vorteil, dass man so feststellen kann, welche Farbe gerade eingeschaltet ist.

Daneben gibt es die Schattenregister, deren Wert alle 1/50 Sekunden basierend auf einem Hardwareregister erneuert werden. Ein Beispiel hierfür sind die Register für Joysticks, Drehregler und Feuerknöpfe, bei denen für die Auslesung der Werte ein spezielles Timing notwendig ist, und deren Werte durch die interne VB-Routine außerdem vorher noch bearbeitet werden (→ Vertikal-Blank-Interrupts).

Während es bei normalen Programmen geradezu unsinnig ist, nicht die vom Betriebssystem bereitgestellten Schattenregister zu verwenden, muss man dies bei DLI-Routinen (→ ANTIC) unbedingt tun, da sonst einerseits die ausgelesenen Werte nicht aktuell sind und andererseits veränderte Werte keine Änderung auf dem Bildschirm bewirken.

2.24 SIO (Serial Input/Output Routine)

Die serielle Ein- und Ausgabe-Routine ermöglicht es, in einer Stufe unter der zentralen Ein- und Ausgaberoutine ("Central Input/Output", → CIO) direkt mit externen Geräten zu kommunizieren. Dabei ist nicht, wie bei der CIO, eine universelle Programmierung für verschiedene Geräte möglich, sondern es muss exakt zwischen den einzelnen Geräten unterschieden werden. Für die verschiedenen Geräte gibt es daher auch verschiedene Kommandos und Treiberroutinen. Im Bereich der Systemvariablen

ist der Bereich von 768 bis 779 ($300-$30B) als "Device Control Block" (DCB) für die SIO reserviert. Der Einsprungpunkt für die SIO ist SIOV (58457; $E459), wohin nach Setzen des DCB gesprungen werden muss. Das DSTATS-Register (771; $303) gibt jeweils an, ob Daten empfangen (dann ist Bit 6 gesetzt) oder gesendet (dann ist Bit 7 gesetzt) werden sollen. Nach jedem Einsprung wird im Y-Register und im DSTATS-Register der Fehlercode zurückgeliefert. Eine Fehlercodetabelle befindet sich im Tabellenteil. Z.B. könnte eine Fehlerabfangroutine wie folgt aussehen:

```
1000 SIOV=$E459
...      ...;Initialisierung der SIO
2000     JSR SIOV
2010     BMI ERROR
2020     ...;Ein-/Ausgabevorgang gelungen
...      ...;weiterer Programmcode
3000 ERROR CPY #$80
3010     BEQ BREAKGEDRUECKT
3020     CPY #...;Hier können die einzelnen
3030             ;Fehlermöglichkeiten geprüft werden.
```

Beim Statuskommando (S: 83; $53) werden die Statusbytes der "intelligenten" Geräte (Drucker, Diskettenstation, RS232-Modul) in DVSTAT (746-749; $2EA-$2ED) abgelegt. Genauere Informationen zu diesen vier Bytes liefert die Speicherübersicht. Im Folgenden nun eine Übersicht, wie der DCB initialisiert werden muss, um mit den einzelnen Geräten zu kommunizieren.

2.25 Diskettenstation

Für die Diskettenstation existiert ein zusätzlicher Vektor (DSKINV: 58451; $E453), der einige von den nachfolgend angegebenen Parametern setzt. Diese Parameter sind durch (‡) gekennzeichnet. Bei den 400/800er Geräten wird das P-Kommando (80; $50), das Kommando also, das das schnelle Schreiben ohne Überprüfung ("Verify") ermöglicht, nicht unterstützt. Um dennoch dieses Kommando verwenden zu können, müssen sämtliche Parameter einschließlich des Kommandos usw. beim Schreiben jedes Sektors neu gesetzt werden. Im Normalfall ist es auf jeden Fall empfehlenswert, den DSKINV-Vektor zu verwenden.

DDEVIC (768; $300): 49 ($31) (‡)
DUNIT (769; $301): 1-8 für die einzelnen Diskettenstationen
DCOMND (770; $302) → Tabelle 2.19

DSTATS (771;$303): vor dem Einsprung in die SIO:

Beim !-, W-, P-Kommando: 128 ($80) (‡)
Beim S- und R-Kommando: 64 ($40) (‡)

Nach der Rückkehr von der SIO:

Y-Register: 1= Status o. k.
>127=Fehler (s. Gesamttabelle)

DBUFLO (772; $304): +
DBUFHI (773; $305): Adresse des zu übertragenden Datenblocks
DTIMLO (774; $306): Timeout-Wert für die Diskettenstation
Normalerweise 7, für das Formatier
kommando 160 ($A0) (‡)

DBYTLO (776; $308): Sektorlänge (‡)
DBYTHI (777; $309): normalerweise 128 ($80) Bytes

DAUX1 (778; $30A): +
DAUX2 (779; $30B): anzusprechende Sektornummer

Nummer Dez (Hex)	**ATASCII**	**Bezeichnung**	**Anmerkung**
33 ($21)	!	Formatieren	40 Spuren mit je 18 Sektoren (720 Sektoren)
34 ($22)	"	Formatieren	40 Spuren mit je 26 Sektoren (1040 Sektoren)
78 ($4E)	N	Konfigurationsblock einlesen	nur mit Laufwerken mit doppelter Schreibdichte
79 ($4F)	O	Konfigurationsblock schreiben	nur mit Laufwerken mit doppelter Schreibdichte
80 ($50)	P	Sektor schreiben	ohne automatische Überprüfung
82 ($52)	R	Sektor lesen	
83 ($53)	S	Status	Statuswert einlesen
87 ($57)	W	Sektor schreiben	mit automatischer Überprüfung

Tab. 2.19: Steuerbefehle für die Diskettenstation

Anmerkung: Das angesprochene Laufwerk muss natürlich das jeweilige Kommando ausführen können; ansonsten sind die Ergebnisse nicht so wie erwartet bzw. Disketten falsch oder fehlerhaft formatiert.

Bemerkungen zum N- und O-Kommando:
Es gibt von verschiedenen Herstellern Diskettenstationen bzw. Erweiterungen dafür, die neben der normalen Schreibdichte (128 Bytes pro Sektor) auch Disketten mit doppelter Schreibdichte, also mit 256 Bytes pro Sektor lesen und schreiben können. Dieser sogenannte ATARI-815-Standard ist der Einzige für Diskettenstationen mit doppelter Schreibdichte. Nun können verschiedene Parameter in der Station verändert werden. Diese Parameter sind im Konfigurationsblock gespeichert. Er kann mit dem N- bzw. O-Kommando gelesen bzw. geschrieben werden.

Der Konfigurationsblock ist zwölf Bytes lang und wie folgt aufgebaut:

Byte	**Bedeutung**
0	Zahl der Spuren (normalerweise 40)
1	Schrittrate (meist 0=6ms, 1=12ms, 2=2ms, 3=3ms)
2,3	Zahl der Sektoren pro Spur
4	Zahl der Schreib-/Leseköpfe -1
5	Aufzeichnungsverfahren (0=FM, 4=MFM)
6,7	Zahl der Bytes pro Sektor
8	Laufwerk aktiv ? (in der Regel $FF)
9	Übertragungsrate
10,11	reserviert

Die Doppelbytes sind nicht in der üblichen 6502-Reihenfolge (LSB/MSB), sondern umgekehrt abgespeichert!

Mit "Schrittrate" wird die Spurwechselgeschwindigkeit bezeichnet. Sie unterscheidet sich von Hersteller zu Hersteller und ist daher nicht sinnvoll anzuwenden.

Im Byte 4 steht normalerweise 0, da von der tatsächlichen Zahl der Köpfe 1 abgezogen wird.

Byte 8 hat keine sinnvolle Anwendung, da mit ihm nur das Laufwerk "Off line" geschaltet werden kann. Um es wieder zu aktivieren, muss man es aus- und einschalten.

Die Übertragungsrate ist wie die Schrittrate nicht einheitlich definiert und somit ebenfalls nicht sinnvoll anzuwenden.

Um einen Konfigurationsblock von Laufwerk 1 in den Computer an die Adresse $600 einzulesen, könnte folgendes Programm verwendet werden:

```
LDA #$31
STA DDEVIC
LDA #1 ;Laufwerk 1 wird angesprochen
STA DUNIT
LDA #'N ;Daten iNto Computer lesen
STA DCOMND
LDA #$40
STA DSTATS
LDA #0
STA DBUFLO
LDA #6 ;an Adresse $600
STA DBUFHI
LDA #$E
STA DTIMLO
LDA #$C ;12 Bytes
STA DBYTLO
LDA #0
STA DBYTHI
JSR SIOV
BMI ERROR ; kein Laufwerk mit doppelter Schreibdichte
```

2.26 Drucker

DDEVIC	(768; $300):	64 ($40)
DUNIT	(769 ; $301):	Bei den 400/800 Geräten wird nur ein Drucker (Wert in diesem Register also eins) unterstützt, die XL/XE-Geräte lassen dagegen 2 Drucker zu.
DCOMND	(770; $302)	S (83; $53): Status W (87;$57): Schreiben
DSTATS	(771; $303):	vor dem Einsprung in die SIO: Beim W-Kommando: 128 ($80) Beim S-Kommando: 64 ($40) Nach der Rückkehr von der SIO: 1 = Status o. k. >127=Fehler (s. Gesamttabelle)
DBUFLO	(772; $304)	
DBUFHI	(773; $305):	Adresse des zu druckenden Textes

DTIMLO (774; $306): Timeout-Wert für den Drucker (30; $1E)

DBYTLO (776; $308):
DBYTHI (777; $309): Länge des zu druckenden Textes

2.27 Kassettenrecorder

Der Kassettenrecorder ist das einzige nicht "intelligente" Gerät für die ATARI-Computer, d.h. es verfügt nicht über einen eingebauten Controller. Da die einzelnen Blöcke auf dem Magnetband verschiedene Kommandos enthalten können, die vom eingebauten Treiber (C:) unterstützt werden, wird empfohlen, diesen ausschließlich zu verwenden. Der Vollständigkeit halber werden hier aber auch die vom Treiber innerhalb des DCB verwendeten Bytes angegeben.

DDEVIC (768; $300): 96 ($60)

DUNIT (769; $301): 0

DCOMND (770; $302): R (82 ; $52): Lesen
W (87; $57): Schreiben

DSTATS (771 ; $303): vor dem Einsprung in die SIO:

Beim W-Kommando: 128 ($80)
Beim R-Kommando: 64 ($40)

Nach der Rückkehr von der SIO:

1 =Status o k.
>127 =Fehler (s. Gesamttabelle)

DBUFLO (772; $304): 253 ($FD)
DBUFHI (773; $305): 3 (Adresse des Kassettenpuffers CASBUF)

DTIMLO (774; $306): Timeout-Wert 35

DBYTLO (776; $308):
DBYTHI (777; $309): 131 ($83) Recordlänge (X)

DAUX1 (778; $30A): 0
DAUX2 (779; $30B): 0= normale Lücken ("Gaps") zwischen den Records
128= kurze Lücken zwischen den Records

2.28 Parallelbusgeräte

Am Parallelbus des ATARI lassen sich diverse Erweiterungen anschließen - unter anderem auch bis zu acht Parallelbusgeräte. Diese können sich mittels eigenem ROM in die Reset-, CIO-, SIO- und/oder IRQ-Routine des Betriebssystems einklinken, wovon dieser Abschnitt handelt.

Für das ROM eines solchen Parallelbusgeräts steht der 2 KB große Bereich $D800 bis $DFFF zur Verfügung, der normalerweise von den Fließkommaroutinen im Betriebssystem-ROM belegt ist. Wenn das Betriebssystem das ROM des Parallelbusgeräts Nummer n (0-7) aktivieren will, setzt es dazu Bit n in PDVS (53759; $D1FF) und dessen Schattenregister SHPDVS (584; $248). Jedes Bit in PDVS identifiziert also eins der acht möglichen Parallelbusgeräte. Wenn nun der ATARI lesend auf $D800 bis $DFFF zugreift, muss das Parallelbusgerät n sein ROM einblenden und die MPD-Leitung (Math pack disable) sowie die EXTSEL-Leitung (External select) auf Low setzen, um das Betriebssystem-ROM und das darunterliegende RAM auszublenden. Um das Geräte-ROM wieder zu deaktivieren, löscht das Betriebssystem Bit n in PDVS wieder, worauf das Gerät sein ROM nicht mehr einblenden darf.

Das Geräte-ROM muss folgenden 28 Byte langen Vorspann haben:

PDCKSM	55296/7	$D800/1	(Prüfsumme über das ROM)
PDREVN	55298	$D802	(Versionsnummer des ROMs)
PDID1	55299	$D803	muss $80 (128) sein
PDTYPE	55300	$D804	(Gerätetyp)
PDIOV	55301-03	$D805-07	JMP-Sprungbefehl zur SIO-Routine des Geräts
PDIRQV	55304-06	$D808-0A	JMP-Sprungbefehl zur IRQ-Routine des Geräts
PDID2	55307	$D80B	muss $91 (145) sein
PDNAME	55308	$D80C	(Geräteidentifikation des CIO-Treibers)
PDVV	55309-22	$D80D-18	Vektortabelle des CIO-Treibers des Geräts
PDINIT	55321-23	$D819-1B	JMP-Sprungbefehl zur Reset-Routine des Geräts

Die Speicherstellen mit in Klammern angegebener Beschreibung haben für das Betriebssystem keine Bedeutung und können daher auch 0 sein. PDID1 und PDID2 dienen zur Prüfung, ob überhaupt ein Parallelbus-Geräte-ROM vorhanden ist, und müssen daher die angegebenen Werte enthalten, sonst ignoriert das Betriebssystem das Geräte-ROM. Daneben sind noch folgende Speicherstellen von Bedeutung:

VPIRQ	568/9	$238/9	Zeiger auf die generische Parallelbus-IRQ-Routine
PDVMSK	583	$247	Bit n aktiviert PDIOV und PDVV im ROM von Gerät n
SHPDVS	584	$248	Schattenregister von PDVS
PDIMSK	585	$249	Bit n aktiviert PDIRQV im ROM von Gerät n
GPDVV	58511	$E48F	Vektortabelle des generischen Parallelbus-CIO-Treibers
GPDINV	58523	$E49B	Ruft die PDINIT-Routinen aller Parallelbus-geräte auf
53504-53758		$D100-$D1FE	Platz für I/O-Register des Geräts
PDVIRQ	53759	$D1FF	Ist Bit n hier und in PDIMASK gesetzt, hat Gerät n einen IRQ ausgelöst. Dieses Register kann nur gelesen werden.
PDVS	53759	$D1FF	Wird Bit n gesetzt, muss Gerät n sein ROM aktivieren. Dieses Register kann nur beschrieben werden.

Das ROM eines Parallelbusgeräts funktioniert folgendermaßen:

RESET

Die Reset-Routine des Betriebssystems ruft (über GPDINV) der Reihe nach die PDINIT-Routinen der Parallelbusgeräte 0 bis 7 per JSR auf. Die Geräte können hier ihre Initialisierung vornehmen und sich für die CIO-, SIO- oder IRQ-Routine des Betriebssystems wie folgt registrieren:

– Jedes Gerät, das sich in die CIO- oder SIO-Routine des Betriebssystems einklinken will, muss hier "sein" Bit in PDVMSK (583; $247) setzen. Das geht am einfachsten mit SHPDVS (584; $248), wo das richtige Bit schon gesetzt ist: LDA PDVMSK, ORA SHPDVS, STA PDVMSK.

– Jedes Gerät, das einen CIO-Treiber bereitstellt, muss dessen Vektortabelle ab PDVV in seinem ROM stehen haben und die Adresse der Vektortabelle des generischen Parallelbus-Treibers GPDVV zusammen mit irgendeiner Geräteidentifikation in die Treibertabelle HATABS (794; $31A) eintragen. Stellt ein Gerät keinen CIO-Treiber bereit, muss es ab PDVV dennoch eine Vektortabelle stehen haben, deren Einträge alle auf CLC, RTS zeigen.

– Jedes Gerät, das IRQs (maskierbare Interrupts) auslöst, muss "sein" Bit in PDIMSK (585; $249) setzen: LDA PDIMSK, ORA SHPDVS, STA PDIMSK.

SIO

Die SIO-Routine (SIOV; 58457; $E459) ruft als Erstes die PDIOV-Routine der in PDVMSK registrierten Geräte in der Reihenfolge 0 bis 7 auf. Eine PDIOV-Routine kann anhand des Gerätetyps DDEVIC (768; $300) und der Gerätenummer DUNIT (769; $301) entscheiden, ob sie die SIO-Funktion ausführen will. Tut sie dies, muss sie danach einen Fehlercode im Y-Register ablegen und sich mit gesetztem Carry-Flag beenden (SEC, RTS), worauf die SIO-Routine sich ebenfalls beendet. Andernfalls muss sich die PDIOV-Routine mit gelöschtem Carry-Flag beenden, worauf die SIO-Routine die PDIOV-Routine des nächsten Geräts aufruft usw. Führt kein Gerät die SIO-Funktion aus, wird zum normalen SIO über den SIO-Anschluss verzweigt. Die PDIOV-Routine darf auch nur DUNIT (769; $301) ändern, um z.B. Zugriffe auf ein anderes Disklaufwerk umzuleiten. Die SIO-Routine stellt am Ende den ursprünglichen DUNIT-Wert wieder her.

PDIOV wird von Festplatten-Interfaces am Parallelbus benutzt, um SIO-Aufrufe auf die Festplatte umzuleiten.

CIO

Wenn die CIO-Routine (CIOV; 58454; $E456) den generischen Parallelbus-Treiber (GPDVV) aufruft, leitet dieser den Aufruf an die im Geräte-ROM abgelegte Vektortabelle PDVV weiter. Ähnlich wie bei der SIO-Routine müssen sich die Treiberroutinen mit gesetztem Carry-Flag beenden, wenn sie die gewünschte CIO-Funktion ausgeführt haben, andernfalls mit gelöschtem Carry-Flag. Führt kein Gerät die CIO-Funktion aus, gibt es einen Error 130.

IRQ

Wenn ein Gerät über die IRQ-Leitung einen IRQ auslöst, muss es dafür sorgen, dass beim lesenden Zugriff auf PDVIRQ "sein" Bit gesetzt ist. Die IRQ-Routine im Betriebssystem prüft, ob gleiche Bits in PDVIRQ und PDIMSK gesetzt sind (LDA PDVIRQ, AND PDIMSK, BNE ...). Ist das der Fall, wird über den Vektor VPIRQ (568/9; $238/9) die generische Parallelbus-IRQ-Routine aufgerufen. Diese ruft die PDIRQV-Routine des Parallelbusgeräts mit der kleinsten Nummer auf, dessen Bit in PDVIRQ und PDIMSK gesetzt ist. Die PDIRQV-Routine kann die A- und X-Register benutzen, ohne sie auf den Stack zu retten, und muss sich mit RTS beenden.

Hier eine allgemeine Vorlage für ein Parallelbus-Geräte-ROM:

```
PDVMSK  = $247        ; zur Aktivierung von PDIOV und PDVV
SHPDVS  = $248        ; PDVS-Schattenregister
PDIMSK  = $249        ; zur Aktivierung von PDIRQV
PHENTV  = $E486       ; Gerät in HATABS eintragen
GPDVV   = $E48F       ; Vektortabelle des generischen
```

```
                        Parallelbus-Treibers

DDEVIC  = $300         ; SIO-Gerätetyp
DUNIT   = $301         ; SIO-Gerätenummer

        .ORG $D800

PDCKSM  .WORD 0
PDREVN  .BYTE 1        ; ROM-Versionsnummer
PDID1   .BYTE $80
PDTYPE  .BYTE 0
PDIOV   JMP PSIO       ; SIO-Routine
PDIRQV  JMP PIRQ       ; IRQ-Routine
PDID2   .BYTE $91
PDNAME  .BYTE 'X       ; Geräteidentifikation des CIO-Treibers
PDVV    .WORD POPEN-1,PCLOSE-1
        .WORD PREAD-1,PWRITE-1    ; falls Gerät einen CIO-
                                    Treiber hat
        .WORD PSTAT-1,PSPEC-1
PDVV    .WORD EXIT-1,EXIT-1
        .WORD EXIT-1,EXIT-1   ; falls Gerät keinen Treiber hat
        .WORD EXIT-1,EXIT-1
PDINIT  JMP PINIT      ; Reset-Routine

PINIT   LDA PDVMSK     ; nur für SIO und CIO-Treiber
        ORA SHPDVS
        STA PDVMSK

        LDA PDIMSK     ; nur für IRQ
        ORA SHPDVS
        STA PDIMSK

        LDX PDNAME     ; nur für CIO-Treiber:
        LDY #<GPDVV ; Generischen PB-Treiber in HATABS
        LDA #>GPDVV ; mit Geräteidentifikation eintragen
        JSR PHENTV

        RTS

EXIT    CLC
        RTS

PSIO    LDA DDEVIC     ; nur für SIO
        CMP #...
        BNE EXIT       ; prüfen, ob SIO-Funktion
```

```
        LDA DUNIT    ; ausgeführt werden soll
        CMP #...
        BNE EXIT
        ...
        SEC
        RTS

PIRQ    ...          ; nur für IRQ
        RTS

POPEN   ...          ; nur für CIO-Treiber
        SEC
        RTS

; analog für PCLOSE, PREAD, PWRITE, PSTAT, PSPEC
```

2.29 Sound

Der Sound wird mit dem POKEY erzeugt, wie nachfolgend beschrieben. Der GTIA verfügt allerdings auch über beschränkte Soundfähigkeiten.

2.29.1 Technische Möglichkeiten

Der ATARI-Computer besitzt vier unabhängig voneinander benutzbare Tonkanäle, die sowohl reine Rechtecktöne als auch Geräusche bzw. Verzerrungen erzeugen können.

Zusätzlich kann jeder einzelne Tonkanal über die Volume-Only-Funktion direkt die Membran des Lautsprechers auslenken, sodass praktisch jede Frequenzkurve erzeugbar ist.

Außerdem können einzelne Tonkanäle als Höhenfilter für andere Kanäle eingesetzt werden.

2.29.2 Einige Begriffsbestimmungen

Welle: Der Begriff "Welle" bezeichnet die an die Lautsprechermembran angelegte Spannung (d. h. deren Auslenkung) in Abhängigkeit zur Zeit. Es gibt im Wesentlichen vier Grundtypen von Wellen.

Die Rechteckwelle: Diese Form der Welle wird vom ATARI-Computer erzeugt und sieht wie folgt aus:

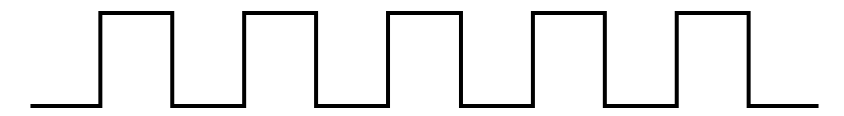


Abb. 2.39: Rechteckwelle

Die Sägezahnwelle: Diese Form der Welle wird vor allem von Blechblasinstrumenten erzeugt und ergibt daher auch einen "blechernen Klang". Sie sieht wie folgt aus:

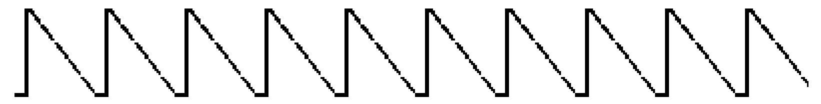


Abbildung 2.1: Sägezahnwelle

Die Sinuskurve: Diese Form der Welle wird z.B. vom Menschen beim Singen erzeugt. Die Frequenzkurve sieht wie folgt aus:



Abb. 2.40: Sinuskurve

Die unregelmäßige Form der Welle: Geräusche haben einen solchen Frequenzverlauf. Eine solche Welle kann z.B. wie folgt aussehen:

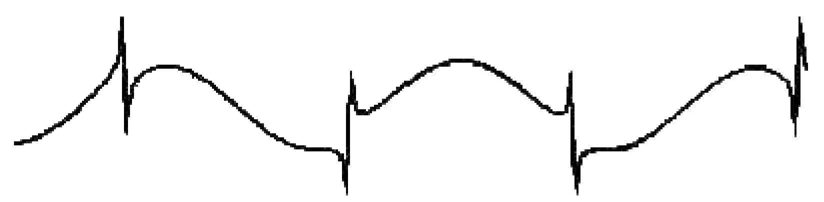


Abb. 2.41: beliebige Welle

Polyzähler/Schieberegister: Diese Register haben eine zentrale Bedeutung für die Geräuscherzeugung, da sie praktisch zufällige Bitfolgen liefern, die dann als "Ursprungswelle" für Geräusche dienen. Das Schieberegister hat eine bestimmte Bit-Breite. Beim ATARI-Computer gibt es 4-, 5- und 17-Bit-Schieberegister. Bekommt das Schieberegister nun einen

bestimmten Befehl, werden alle Bits um eine Position nach rechts geschoben. Nehmen wir als Beispiel ein 4-Bit-Schieberegister: Das Bit 4 wandert an die Stelle von Bit 3, Bit 3 an die Stelle von Bit 2 usw. Bit 1 schließlich steht als Endwert zur Verfügung.

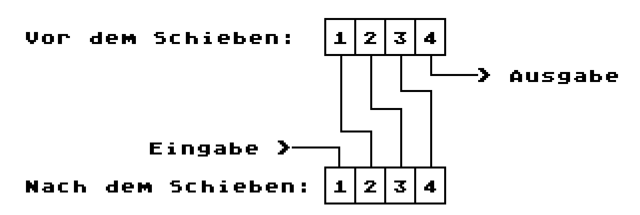


Abb. 2.42: Schieberegister

Nun muss schließlich ein neues Bit an die Stelle von Bit 4 treten. Werden z.B. zwei Bit des Schieberegisters miteinander exklusiv oderiert und das entstehende Bit als Bit 4 genommen, so ist die Wiederholungsrate der entstehenden Schwingung von der Größe des Schieberegisters abhängig; je größer die Bit-Breite des Schieberegisters ist, desto seltener wird die gleiche Bitfolge als Schwingung erzeugt, d. h. desto unregelmäßiger und zufälliger ist die Schwingung. Diese Art der Bitverarbeitung bezeichnet man als Polyzähler. Eine weitere Skizze macht dessen Funktionsweise deutlich.

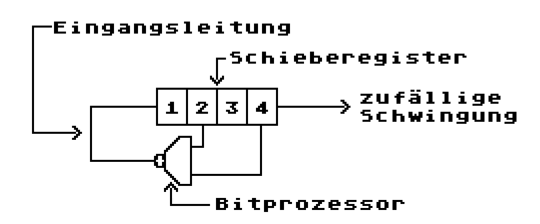


Abb. 2.43: Polyzähler

2.29.3 Für die Tonerzeugung wichtige Register

AUDF1-4 (53760, 53762, 53764, 53766; $D200, $D202, $D204, $D206):
Diese vier Register sind die sogenannten "AUDio-Frequency"-Register. In ihnen werden die Werte für die Frequenz des zu erzeugenden Tones abgespeichert.

AUDC1-4 (53761, 53763, 53765, 53767; $D201, $D203, $D205, $D207):
In diesen vier "AUDio-Control"-Registern werden die Daten für die Verzerrung und Lautstärke des Tones abgelegt.

AUDCTL (53768; $D208):
Dieses "AUDio-ConTroL"-Register ermöglicht eine umfassende Kontrolle über das Zusammenwirken der einzelnen Tonkanäle (z.B. Zusammenschalten zweier Tonkanäle oder Einsatz eines Tonkanals als Höhenfilter für einen anderen). Um normale Töne zu erzeugen, muss 0 in diesem Register abgelegt werden.

SKCTL (53775; $D20F)
Schreibt man den Wert 3 in dieses Register, so wird der POKEY-Chip initialisiert.

Die Initialisierungssequenz in Maschinensprache sieht wie folgt aus:

```
LDA #3
STA $D20F
LDA #0
STA $D208
```

2.29.4 Benutzung der einzelnen Register

Zunächst muss man davon ausgehen, dass als Taktfrequenz eine Frequenz von 64 kHz (kHz=Kilohertz, 1000 Schwingungen pro Sekunde) zur Verfügung steht. Diese Taktfrequenz wird jeweils durch den Wert N in den AUDF-Registern geteilt, d.h., es wird nur jeder N-te Impuls ausgegeben. Dieser Wert N kann zwischen 0 und 255 ($FF) liegen. Da eine Division durch 0 selbstverständlich nicht möglich ist, addiert der POKEY-Chip automatisch 1 hinzu. Die Werte für N rangieren also zwischen 1 und 256 ($100). Da der POKEY-Chip vor der Ausgabe des Signals zum Fernseher die Frequenz des Signals (Rechteckform oder verzerrt) noch einmal durch zwei teilt, sind im Normalfall also Schwingungen im Bereich von 32 kHz bis hinunter zu 125 Hz möglich.

Folgende Formel gibt diesen Sachverhalt wieder:

$$\text{Ausgangsfrequenz} = \frac{\text{Taktfrequenz}}{\text{N (Wert in AUDF plus 1)}}$$

Wird eine Verzerrung des Signals gewünscht, so wird durch Verwendung der Polyzähler ein Zufallsfaktor in das Signal gebracht, d.h., die Form des

Signals wird verändert. Diese Verzerrung wird über die AUDC-Register kontrolliert. Die einzelnen Bits haben dabei die folgende Funktion:

Bit	Funktion
0-3	Durch diese vier Bits wird die Lautstärke festgelegt. Sie kann Werte zwischen 0 und 15 annehmen. Die Gesamtlautstärke aller vier Tonkanäle sollte nicht größer als 32 ($20) sein, da sonst die Tonausgabe verzerrt klingt.
4	Volume-Only-Funktion ein/aus (Erklärung siehe unten)
5-7	Diese Bits kontrollieren die Verzerrung nach der Tabelle 2.20.

Bit	7	6	5	Wirkung auf die Ausgangsfrequenz
	1	1	0	Verzerrung durch 4-Bit-Polyzähler
	1	X	1	Reine Rechteckkurve
	1	0	0	Verzerrung durch 17-Bit-Polyzähler
	0	1	0	Verzerrung durch 5-Bit-Polyzähler, anschließend durch 4-Bit-Polyzähler
	0	X	1	Verzerrung durch 5-Bit-Polyzähler
	0	0	0	Verzerrung durch 5-Bit-Polyzähler, anschließend durch 17-Bit-Polyzähler

Tab. 2.20: Verzerrungstabelle

Das Zeichen 'X' bedeutet, dass der Zustand dieses Bits nicht relevant ist.

Vor einer Verzerrung wird die Taktfrequenz durch den Wert in AUDF geteilt, nachher noch einmal durch zwei. Man kann sagen, je mehr und je breitere Polyzähler auf die Ausgangsfrequenz angewandt werden, desto zufälliger ist das ausgegebene Signal. Wird eine Verzerrung mit dem 17-Bit-Polyzähler durchgeführt, kommt das erzeugte Signal dem sogenannten "weißen Rauschen", das ein absolut zufälliges Signal meint, recht nahe. Die Tabelle 2.21 gibt in etwa die akustischen Entsprechungen für die Verzerrung durch die verschiedenen Bits in AUDC an.

Bit 7	6	5	Wirkung in den verschiedenen Frequenzbereichen		
			Hohe Freq.	Mittlere Freq.	Tiefe Freq.
1	1	0	Rasierer	Mofa	Flugzeug
1	X	1	Keine Verzerrung	Rechteckkurve	
1	0	0	Wasserfall	Radioknistern	Einstürzende Bauten
0	1	0	Knattern von Autos		Lagerfeuer
0	X	1	Trafo Elektromotor		Maschinengewehr
0	0	0	Wind Zischen	Kaminfeuer	Geigerzähler

Tab. 2.21: Tabelle der Geräuschbeschreibungen

Das Zeichen 'X' bedeutet, dass der Zustand dieses Bits nicht relevant ist.

Nun zur "Volume-Only"-Möglichkeit: Ist Bit 3 eines AUDC-Registers gesetzt, so wird eine Spannung an den Lautsprecher angelegt, die proportional zum Lautstärkewert in diesem AUDC-Register ist. Sowohl der Verzerrungswert als auch der zugehörige AUDF-Wert werden ignoriert. Die Lautsprechermembran lässt sich also in sechzehn Stellungen auslenken. Die Stellung der Membran bleibt bis zur nächsten Änderung erhalten. Diese stark von der passenden Software abhängigen "Volume-Only"-Möglichkeit erlaubt es, praktisch alle Toneffekte zu programmieren. Sprachsynthesizer, die durch Software betrieben werden, erzeugen die Sprache über nur einen Tonkanal mit der "Volume-Only"-Möglichkeit. Da es im Wesentlichen auf die Programmgeschwindigkeit ankommt, ist es günstig, alle Interrupts und den ANTIC auszuschalten. Das nachstehende Assemblerprogramm ermöglicht es, verschiedene Frequenzkurven auszuprobieren. Verschiedene Tonhöhen können über die Tastatur eingegeben werden. Das Programm soll nur zur Verdeutlichung dieser "Volume-Only"-Möglichkeit dienen und kann natürlich noch stark verbessert werden.

```
1000      *= $4000 ;beliebige Startadresse
1010      JMP START
1020 ;
1030 ;Festlegung der Labelnamen
1040 ;
1050 AUDC1 = $D201
1060 AUDCTL = $D208
1070 KBCODE = $D209
1080 IRQEN = $D20E
1090 SKCTL = $D20F
1100 DMACTL = $D400
```

```
1110 VCOUNT = $D40B
1120 NMIEN = $D40E
1130 ;
1140 ;Festlegung der Variablen und Tabellen
1150 ;
1160 LEN =   DTAB-VTAB
1170 ;
1180 ;Es folgt die Tabelle der Frequenzkurve
1190 ;31: Membran ganz ausgefahren
1200 ;16: Membran ganz eingezogen
1210 ;Die Laenge der Tabelle ist beliebig
1220 ;(bis 255 Werte)
1230 ;
1240 VTAB .BYTE 31,28,25,22,19
1250 DTAB .DS 256
1260 ;
1270 ;Nachstehend die Tabelle für die
1280 ;Lautstaerke der einzelnen Frequenz-
1290 ;Kurvendurchlaeufe
1300 ;Werte zwischen 0 (ganz leise) und 4
1310 ;(ganz laut) sind moeglich. Auch hier ist die
1320 ;Laenge der Tabelle frei waehlbar.
1330 ;
1340 VOLTAB .BYTE 1,3,4,4,4,3,3,2,2,1
1350 T   .BYTE 0
1360 TABLEN = T-VOLTAB
1370 ;
1380 ;Es folgen verschiedene Variablen.
1390 ;
1400 TEMPO .BYTE 1
1410 VOL .BYTE 0
1420 A1  .BYTE 0
1430 A2  .BYTE 0
1440 A3  .BYTE 0
1450 WERT .BYTE 0
1460 TEMPO1 .BYTE 0
1470 REG .BYTE 0
1480 COUNT .BYTE 0
1490 KEYFLAG .BYTE 0
1500 ;
1510 ;Ab hier startet das Programm
1520 ;
1530 START LDA #0
1540     STA AUDCTL
1550     STA NMIEN ;Alle Interrupts
```

```
1560      STA IRQEN  ;werden ausgeschaltet
1570      STA DMACTL
1580      TAX
1590      LDA #3      ;Initialisierung des
1600      STA SKCTL   ;POKEY-Chips
1610 ;
1620 HAUPTSCHLEIFE
1630      LDA COUNT
1640      BNE OK4
1650      LDA SKCTL
1660      CMP #255
1670      BEQ RESET
1680      LDA KEYFLAG
1690      BNE LOOK
1700      LDA SKCTL
1710      CMP #255
1720      BNE SET
1730      RESET LDA #0
1740      STA KEYFLAG
1750 ;
1760 LOOK LDY KBCODE ;Tastaturwert dient
1770      STY COUNT   ;als Zaehlerwert
1780      CPY TEMPO1
1790      BEQ CONT
1800      STY TEMPO1
1810      JMP CO
1820 ;
1830 SET STA KEYFLAG
1840 CO   LDA #0
1850      STA TEMPO
1860      LDY KBCODE
1870      STY COUNT
1880      JMP HAUPTSCHLEIFE
1890 ;
1900 CONT LDA DTAB,X
1910 L1   STA AUDC1
1920 OK4 DEC COUNT
1930      JSR VOLCHECK
1940      LDA COUNT
1950      BNE HAUPTSCHLEIFE
1960      INX
1970      CPX #LEN
1980      BNE HAUPTSCHLEIFE
1990      LDX #0
2000      BEQ HAUPTSCHLEIFE
```

```
2010 ;
2020 PUSHREG STA A1
2030     STX A2
2040     STY A3
2050     RTS
2060 ;
2070 PULLREG LDA A1
2080     LDX A2
2090     LDY A3
2100     RTS
2110 ;
2120 VOLCHECK LDA VCOUNT
2130     BNE RETU
2140     JSR PUSHREG
2150     INC TEMPO
2160     LDY TEMPO
2170     CPY #TABLEN
2180     BCC OK3
2190     LDY #T-VOLTAB
2200     STY TEMPO
2210 OK3 LDA VOLTAB,Y
2220     STA VOL
2230     LDY #LEN
2240 LOOP LDA VTAB-1,Y
2250     SEC
2260     SBC #16
2270     STA REG
2280     LDA #4
2290     SEC
2300     SBC VOL
2310     TAX
2320     LDA REG
2330     CPX #0
2340     BEQ NODEC
2350 LOOP1 LSR A
2360     DEX
2370     BNE LOOP1
2380 NODEC CLC
2390     ADC #16
2400     STA DTAB-1,Y
2410     DEY
2420     BNE LOOP
2430     JSR PULLREG
2440 RETU RTS
2450 ;
```

Zur Funktion von AUDCTL lässt sich Folgendes sagen: Dieses Byte ermöglicht eine noch weitergehendere Kontrolle über die Tonerzeugung als die vier AUDC-Register. Jedes Bit dieses Bytes hat, falls es gesetzt ist, eine spezielle Bedeutung. Die nachstehende Tabelle macht dies deutlich:

Bit	Dez	Hex	Funktion
7	128	$80	Der 17-Bit-Polyzähler ist nur noch neun Bits lang.
6	64	$40	Kanal 1 wird statt mit 64 kHz mit 1,77 MHz getaktet.
5	32	$20	Kanal 3 wird statt mit 64 kHz mit 1,77 MHz getaktet.
4	16	$10	Die Kanäle 1 und 2 werden als 16-Bit-Register verwendet.
3	8	$08	Die Kanäle 3 und 4 werden als 16-Bit-Register verwendet.
2	4	$04	Kanal 1 gibt nur noch die Frequenzen aus, die höher als die von Kanal 3 liegen. ("High-Pass-Filter")
1	2	$02	Kanal 2 gibt nur noch die Frequenzen aus, die höher als die von Kanal 4 liegen. ("High-Pass-Filter")
0	1	$01	Die Taktfrequenz wird von 64 kHz auf 15 kHz verringert.

Tab. 2.22: Bedeutung der einzelnen Bits von AUDCTL

Zur Erklärung der 16-Bit-Auflösung: Die Abstufungen der einzelnen Töne untereinander werden verringert. Statt 256 Abstufungen stehen nun 65536 Abstufungen zur Verfügung. Kanal zwei bzw. vier werden als höherwertiges Byte, Kanal 1 bzw. 3 als niederwertiges Byte eingesetzt. Die Lautstärke kommt von Kanal 2 bzw. 4. Die Lautstärke von Kanal 1 bzw. 3 muss auf 0 gesetzt werden.

Wird der 17-Bit-Polyzähler auf neun Bits verkürzt, so werden alle Verzerrungen, die den bisherigen 17-Bit-Polyzähler verwenden, natürlich regelmäßiger, nähern sich also denen der 4-und 5-Bit-Polyzähler an.

2.29.5 Berechnung der Tonfrequenzen

Bei den genauen Tonfrequenzen gibt es kein richtig oder falsch, da der Kammerton international nicht eindeutig festgelegt ist. Er liegt je nach Land und Gewohnheiten zwischen 440 und 444 Hz. Meist sind es 440 Hz, in deutschen Sinfonieorchestern sind es z.B. 443 Hz. Daraus folgt, dass man ausgehend von den Basisfrequenzen, die bei NTSC und PAL leicht unterschiedlich sind, die Tonfrequenzen für die Stimmung des eigenen ATARI-Orchesters selbst festlegen muss.

Die Frequenz f der Töne ergibt sich je nach eingestellter Basisfrequenz b in AUDCTL=$D208 aus dem AUDF-Wert x bei PAL zu:

b=1,77 MHz → f=1773447 Hz/2/(7+x)
b=64 kHz → f=63337,4 Hz/2/(1+x) → (1/28 von 1,77 MHz)
b=15 kHz → f=15556,55 Hz/2/(1+x) → (1/114 von 1,77 MHz)

Bei NTSC betragen die Basisfrequenzen 1.789.790 Hz, 63.921,07 Hz und 15.699,91 Hz.

Mit der normalen Basisfrequenz 64 kHz ergeben sich bei PAL-ATARIs folgende AUDF-Werte für reine Töne (Verzerrung=10, A3=443 Hz):

Oktave	C	C#	D	D#	E	F	F#	G	G#	A	A#	H
2	239	226	213	201	190	179	169	159	150	142	134	126
3	119	112	106	100	94	89	84	79	75	70	66	63
4	59	56	53	50	47	44	42	39	37	35	33	31
5	29	27	26	24	23	22	20	19	18	17	16	15
6	14	13	(12)	(12)	11	10		9		8		7

Tab. 2.23: Tontabelle für PAL-Geräte

Die von ATARI offiziell für NTSC-ATARIs festgelegten reinen Töne ergeben sich aus folgenden AUDF-Werten (NTSC, A3=443 Hz):

Oktave	C	C#	D	D#	E	F	F#	G	G#	A	A#	H
2	243	230	217	204	193	182	173	162	153	144	136	128
3	121	114	108	102	96	91	85	81	76	72	68	64
4	60	57	53	50	47	45	42	40	37	35	33	31
5	29	27	26	24	23	22	20	19	18	17	16	15

Tab. 2.24: Tontabelle von ATARI für NTSC-Geräte

Die NTSC-Tonwerte werden in den meisten kommerziellen Programmen verwendet.

2.30 Speicheraufteilung

Wenn man sich die Leistungsdaten des ATARI 130XE durchliest, dann sollte einem eigentlich Folgendes auffallen:

RAM:	128 KByte
BASIC:	8 KByte
Betriebssystem:	14 KByte
I/O Chips:	2 KByte
Selbsttest:	2 KByte
Gesamt:	154 KByte

Allerdings verfügt der Mikroprozessor des ATARI XL/XE, der "Sally"-Chip (vollständig software-, aber nicht pinkompatibel zum 6502), nur über einen 16 Bit breiten Adressbus und kann deshalb eigentlich nur 2^{16} Bytes = 65536 Bytes = 64 KByte ansteuern. Um nun wirklich den gesamten Speicher nutzen zu können, muss zwischen verschiedenen "Bänken" hin- und hergeschaltet werden, d.h., dass insgesamt immer nur gleichzeitig 64 KByte, abwechselnd aber prinzipiell beliebig viel Speicher angesprochen werden kann.

Um die interne Aufteilung des Arbeitsspeichers des ATARI vollständig verstehen zu können, muss man die "Evolution", die im 130XE gipfelt, in Betracht ziehen.

Die ersten beiden Geräte, der ATARI 400 und 800, sind ohne Probleme nur bis zu einem RAM-Speicher von 48 KByte RAM ausbaubar. Dieser Speicher liegt im Bereich von 0 ($0000) bis 49151 ($BFFF). Die folgenden 4096 Bytes (49152-53247; $C000-$CFFF) waren bei diesen Geräten unbenutzt. Die nächsten 2048 Bytes (53248-55295; $D000-$D7FF) sind für die Spezialchips vorgesehen (→ Hardwareregister). Im Bereich von 55296-57343 ($D800-$DFFF) liegen die Fließkommaroutinen und der Rest des Speichers (57344-65535; $E000-$FFFF) wird vom eigentlichen Betriebssystem belegt.

Auch bei den ersten Geräten gab es schon eine Erweiterung des normalen Speicherbereiches durch Umschalten von Speicherbänken (Bank Switching): In dem Moment, in dem eine Cartridge eingesteckt wird, ist das im selben Speicherbereich liegende RAM abgeschaltet (→ Cartridges). Das ATARI-BASIC-Modul belegte beispielsweise den Speicherplatz zwischen 40960 ($A000) und 49151 ($BFFF). Die Möglichkeit des Bank Switching wurde beim ATARI 800 gleichfalls von Drittherstellern zum Einbinden von Erweiterungsspeicher von bis zu 1 MByte genutzt.

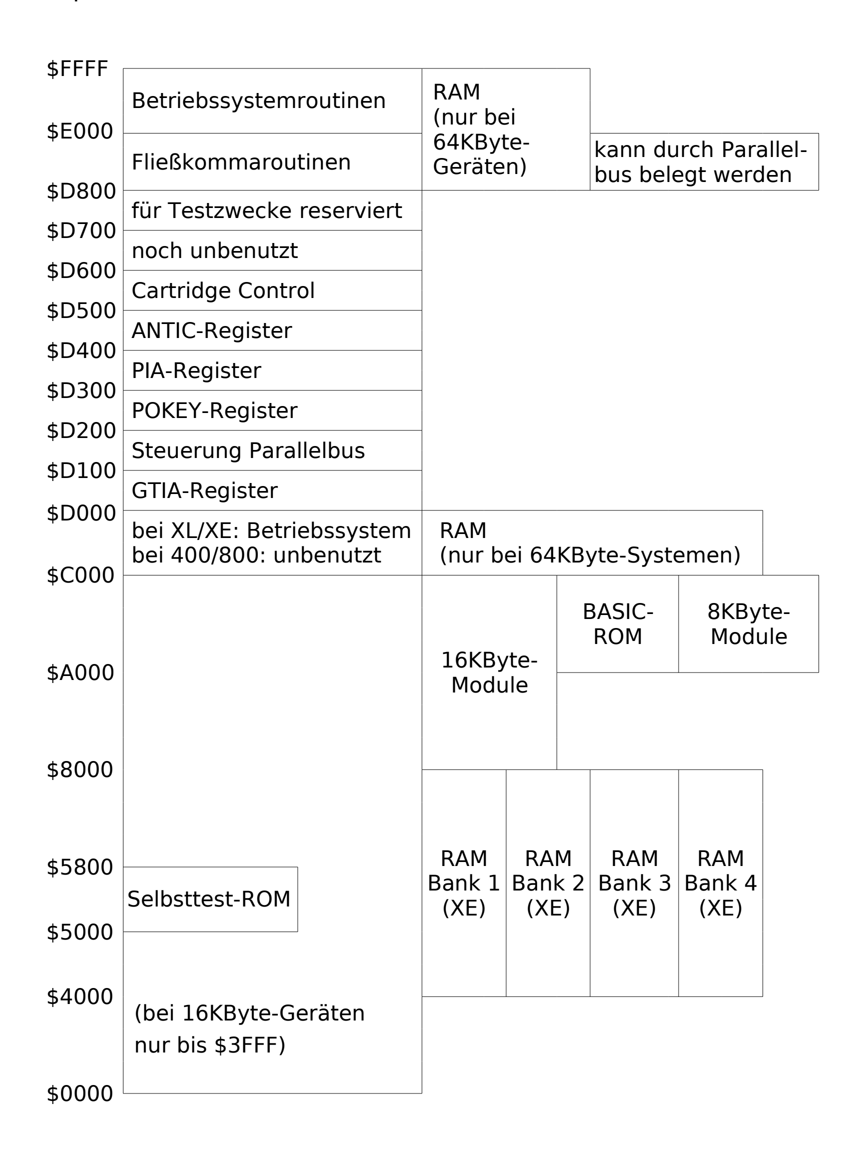


Abb. 2.44: Speicheraufteilung

Damit sind wir schon bei der ersten Ursache, die den Speicheraufbau der neueren Geräte beeinflusst hat. Bei den XL/XE-Modellen ist das ATARI-BASIC, das es vorher als einzelnes Steckmodul gab, eingebaut. Um die

Kompatibilität zu wahren (und nicht nur deshalb), belegt es natürlich noch den gleichen Speicherbereich.

Außerdem verfügen alle neuen Geräte (bis auf den 600XL, solange er nicht auf 64 KByte RAM erweitert ist, z.B. mit dem ATARI 1064) über 16 KByte RAM, die unter dem Betriebssystem "versteckt" liegen. Dies hat den Vorteil, dass man so einfach das Betriebssystem in das RAM übertragen und dort verändern kann. Innerhalb des Speicherbereiches des Betriebssystems liegt das Selbsttestprogramm, und zwar im Bereich von 53248-55295 ($D000-$D7FF). Da dieser Bereich aber stets von den Spezialchips (→ Hardwareregister) benötigt wird, wird er im Bereich von 20480-22527 ($5000-$57FF) eingeblendet.

Kommen wir zum 130XE, der nochmals zusätzlich über 64 KByte RAM verfügt. Dieser Speicherbereich ist in vier Abschnitte unterteilt, die jeweils in 16384-32767 ($4000-$7FFF) eingeschaltet werden. Hier hat sich ATARI allerdings noch etwas einfallen lassen: Es lässt sich nämlich festlegen, ob der Mikroprozessor oder/und der ANTIC (Grafikprozessor) auf den Speicher Zugriff haben. Damit kann der 130XE eigentlich doch auf mehr als 64 KByte, nämlich auf 80 KByte, gleichzeitig zugreifen. Dies gilt ebenso für Speichererweiterungen vom Typ "CS" in allen XL/XE-Geräten.

Da bei XL/XE-Geräten mit 64KByte Speicher die entsprechenden Bits im Port B des PIA (→ 54017; $D301) seitens ATARI ungenutzt blieben, haben findige Entwickler hier eigene Speichererweiterungen in das System eingebunden. Damit wurde die Nutzung von Port B des PIA zum Standard für die Verwaltung von Erweiterungsspeicher bei den XL/XE-Modellen.

Zusätzlich benutzt das Betriebssystem (nur beim XL und XE!) noch einige Hilfsregister, um den Speicher zu verwalten.

BASICF (1016; $3F8) ist ein Flag, das anzeigt, ob das BASIC eingeschaltet ist. Es wird unter anderem abgefragt, wenn ein Warmstart stattfindet, um das BASIC wieder auf den vorherigen Zustand zu schalten. Ist BASIC eingeschaltet, hat BASICF den Wert 0, ansonsten 1.

Über TRIG3 (53267; $D013) wird abgefragt, ob ein Modul eingesteckt ist. Hierbei signalisiert eine 1, dass eine Cartridge eingesteckt (und aktiv) ist, und 0 den gegenteiligen Fall.

GINTLK (1018; $3FA) ist zu diesem Register das Schattenregister.

CARTCK (1003; $3EB) beinhaltet die Prüfsumme der Bytes zwischen 49136-49391 ($BFF0-$C0EF).

Die beiden letzten Register dienen auch dazu, bei einem Warmstart festzustellen, ob ein Modul eingesteckt, herausgezogen oder gewechselt worden ist. In einem solchen Fall würde dann ein Kaltstart ausgeführt.

Zur Steuerung einer Speicherverwaltung benötigt der ATARI natürlich einen Ein- und Ausgabebaustein. Es handelt sich dabei um den Teil der PIA, der beim ATARI 400/800 den dritten und vierten Joystickanschluss steuert. Das Steuerregister ist PORTB (54017; $D301). Schon bevor ATARI beim 130XE eine Speicherverwaltung per Bank Switching einführte, gab es für die bis dahin erschienenen ATARI-Modelle bereits andere Lösungen durch Drittanbieter, die nach dem gleichen Prinzip funktionierten. Für die XL/XE-Modelle wurde eine Vielzahl an Speichererweiterungen bis zu einer Größe von 4 MByte entwickelt. Auch für die erste Generation der 400/800-Modelle gibt es Speichererweiterungen auf bis zu 1 MByte. Da diese bei uns jedoch nicht angeboten werden, soll sich die weitere Betrachtung auf die XL/XE-Modelle beschränken.

Die Belegung der einzelnen Bits bei einem nicht aufgerüsteten 130XE, also der Quasi-Standard:

Bit	Zustand	Wirkung
Bit 7:	an	→ $5000-$57FF RAM
	aus	→ $5000-$57FF Selbsttest (nur wenn Bit 0 an ist)
Bit 6:	(noch unbenutzt)	
Bit 5:	an	→ Zugriff des ANTIC auf Zusatz-RAM abgeschaltet
	aus	→ Zugriff des ANTIC auf Zusatz-RAM angeschaltet
Bit 4:	an	→ Zugriff der CPU auf Zusatz-RAM abgeschaltet
	aus	→ Zugriff der CPU auf Zusatz-RAM angeschaltet
Bit 3+2:	00	→ Bank 1
	01	→ Bank 2
	10	→ Bank 3
	11	→ Bank 4
Bit 1:	an	→ $A000-$BFFF RAM (falls kein Modul aktiv)
	aus	→ $A000-$BFFF BASIC (falls kein Modul aktiv)
Bit 0:	an	→ $C000-$CFFF und $E000-$FFFF Betriebssystem
	aus	→ $C000-$CFFF und $E000-$FFFF RAM

Bei den verschiedenen Schaltungskonzepten werden unterschiedliche Bits verwendet, sodass kein absolut einheitlicher Standard für die Nutzung von Port B des PIA existiert. Allen gemeinsam ist die Nutzung von Bit 2 und 3. Die restlichen Bits werden in den verschiedensten Kombinationen benutzt. Zwei Typen haben sich dabei etabliert: "Rambo" bzw. "Compy-Shop".

Der Typ "Rambo" basiert auf einem Entwurf von Claus Buchholz (The Quarter-Meg ATARI, Dezember 1984) für den 800XL, bei dem ursprüng-

lich 8 Bänke zu je 32 KByte im Bank-Switching-Verfahren genutzt wurden. Mit Erscheinen des 130XE im Frühjahr 1985 wurde von ATARI der neue Standard gesetzt und C. Buchholz passte seine Schaltung für den 800XL auf 16 KByte große Bänke an. Damit ist seine Schaltung zwar XE-kompatibel, weist aber einen gravierenden Unterschied auf: Der getrennte Zugriff von CPU und ANTIC auf das Zusatz-RAM ist nicht möglich. Im Laufe der Jahre etablierte sich der Begriff "Typ Rambo" für Speichererweiterungen, bei denen CPU und ANTIC nur gleichzeitig Zugriff haben.

Für Speichererweiterungen, die den getrennten Zugriff von CPU und ANTIC auf das Zusatz-RAM ermöglichen, wird der Begriff "Typ Compy-Shop" (kurz: CS) verwendet.

Im Laufe der Zeit haben sich primär folgende Nutzungen für den "zusätzlichen" Arbeitsspeicher ergeben:

Der Bereich "unter" dem Betriebssystem wird meist dazu verwendet, entweder das bestehende Betriebssystem zu verändern oder ein anderes zu benutzen. Daneben gibt es auch mehrere DOS-Versionen, die diesen Speicherbereich benutzen und damit im normalen Speicherbereich mehr Platz lassen. Auch Programmiersprachen wie z.B. "Turbo BASIC 1.5 XL" nutzen den Speicher unter dem Betriebssystem.

Die vier zusätzlichen RAM-Bänke des 130XE sind unter DOS 2.5 recht effektiv als RAM-Disk benutzt (→ DOS), was auch bei fast allen anderen DOS der Fall ist. Wer allerdings mehr Speicher als die zusätzlichen 64 KByte des 130XE nutzen will, muss auf DOS von Drittherstellern ausweichen.

Eine weitere Anwendung ist beispielsweise das Umschalten zwischen verschiedenen Seiten mit hochauflösender Grafik.

Daneben benötigt das Betriebssystem auch verschiedene Teile der unteren Speicherregionen. Die erste Seite des Speichers (bei einer Seite handelt es sich stets um einen Speicherbereich, bei dem die obere Hälfte der Adresse gleich ist), also der Bereich von 0-255 ($0-$FF), ist in zwei unterschiedliche Teile unterteilt. Die erste und damit untere Hälfte wird praktisch vollständig vom Betriebssystem benutzt. Der zweite Teil ist im Prinzip für Anwenderprogramme frei. Dabei muss man allerdings beachten, dass, da vom eigentlichen Betriebssystem abgetrennt, auch das BASIC und die Fließkommaroutinen als Anwenderprogramme gelten und daher diesen Teil des Speichers benutzen. Genauere Angaben hierzu finden sich im Speicherplan. Der Bereich von 256-511 ($100-$1FF) wird vom Stapel (Stack) des Mikroprozessors belegt. Weiterhin benötigt das Betriebssystem für Zeiger, Statusregister, Puffer und Ähnliches den Bereich

bis 1151 ($47F). Der restliche Platz bis zum Ende der sechsten Speicherseite (1152-1791; $480-$6FF) ist wiederum frei für Anwenderprogramme. Auch hier gilt wieder, dass beispielsweise die Fließkommaroutinen einen Teil dieses Bereiches verwenden.

2.31 System Timer

Das Betriebssystem des ATARI unterstützt fünf verschiedene Systemtimer. Alle fünf verfügen über einen 16-Bit-Zähler, der jede 1/50 Sekunde vermindert wird. Dabei wird lediglich der erste Timer während des Immediate-VB (→ Vertical Blank Interrupt) verändert. Die ersten beiden Timer verfügen über einen Vektor, der auf das Unterprogramm zeigen muss, das nach Ablauf der im Zähler angegebenen Zeit aufgerufen werden soll. Die drei übrigen Timer verfügen dafür über jeweils ein Flag, das anzeigt, dass der Zähler 0 erreicht hat. Das Anwenderprogramm muss dann selbsttätig darauf reagieren.

Die Zählerregister liegen in den Speicherstellen ab 536 ($218), die Vektoren und Flags ab 550 ($226) (→ Speicherplan).

2.32 Vertikal Blank Interrupt

Der ATARI verfügt über eine Vielzahl von Interrupts (das sind Unterbrechungen des normalen Programmablaufs). Interrupts sind eine sehr nützliche Sache, da man mithilfe eigener Interrupt-Programme praktisch zwei Programme gleichzeitig ablaufen lassen kann, mithin in beschränkter Form eine Art von Multi-Tasking (gleichzeitiger Ablauf mehrerer Programme) möglich ist.

Der Vertikal-Blank-Interrupt (im folgenden VBI oder VB genannt) findet jeweils zwischen der Erzeugung zweier Bilder statt. Auf PAL-Geräten passiert das alle 1/50 Sekunden, auf NTSC-Geräten alle 1/60 Sekunden. Dadurch eignet sich der VBI hervorragend für alle Programme, die in einem festen Rhythmus und ununterbrochen abgearbeitet werden müssen, wie z.B. Action-Spiele. Des Weiteren ist es sehr praktisch, bei Grafikprogrammen alle Änderungen von Hardware-Registern (dazu gleich mehr) im VBI zu unternehmen, da so Synchronisationsprobleme mit dem Bildaufbau, die meistens zu einem unschönen Zucken führen, vermieden werden können.

Das Betriebssystem beinhaltet eine eigene VBI-Routine, die vom Benutzer sowohl ergänzt als auch abgeschaltet werden kann. Im Folgenden eine Beschreibung der Funktionen dieser VBI-Routine:

Zunächst wird die interne Uhr (RTCLOK, 18-20; $12-$14) um eins erhöht. Dabei ist übrigens das letzte der drei Bytes das niederwertigste. Jedes Mal, wenn RTCLOK +1 erhöht wird, also alle 256/50 s, was ungefähr fünf Sekunden entspricht, wird auch das Register ATRACT erhöht. Hat es den Wert 127 ($7F) überschritten, beginnt der bekannte automatische Farbwechsel auf dem Bildschirm, der das Einbrennen in die Bildröhre verhindern soll, wenn für längere Zeit keine Taste gedrückt wird. Zurückgesetzt wird ATRACT entweder durch Drücken einer Taste oder durch das eigene Programm, was wohl immer dann sinnvoll ist, wenn die Tastatur überhaupt nicht verwendet wird (z.B. bei Spielen, Demonstrationsprogrammen etc.).

Daraufhin wird der erste der System-Timer (CDTMV1, 536,537; $218,$219) um 1 erniedrigt (wenn er nicht 0 war). Hat dieses Register den Wert 0 erreicht, wird über den Vektor des Timers (CDTMA1, 550,551; $226,$227) das dazugehörige Unterprogramm aufgerufen, das mit einem "RTS"-Kommando enden sollte.

Als Nächstes wird überprüft, ob CRITIC (66; $42) einen Wert ungleich 0 enthält. In diesem Fall, oder wenn sich das unterbrochene Programm sowieso schon im Interrupt-Modus befand, d.h. das I-Statusbit des 6502-Prozessors gesetzt ist, wird an dieser Stelle die interne VBI-Routine abgeschlossen. Das ist immer dann der Fall, wenn gerade ein zeitkritischer Ein-/Ausgabevorgang (z.B. über Diskettenstation, Kassettenrecorder oder Drucker) stattfindet. In diesem Fall werden also nur die interne Uhr, der erste Timer und das Byte ATRACT verändert.

Bei XL/XE-Geräten folgt nun die Überprüfung, ob seit dem letzten VBI ein Modul eingesteckt worden ist. Dazu wird der Wert von TRIG3 (53267; $D013) mit dem Wert GINTLK (1018; $3FA) verglichen. Wenn ein Unterschied festgestellt wird, beginnt die Abarbeitung einer endlosen Schleife, die nur durch Drücken von RESET unterbrochen werden kann.

Es folgt die Übertragung einiger Werte zwischen "Schattenregister" und tatsächlichem Register in einem der Spezialchips (→ Schattenregister).

Hier eine Aufstellung der übertragenen Bytes:

PENV	(54285; $D40D)	→	LPENV	(565; $235)
PENH	(54285; $D40C)	→	LPENH	(564; $234)
SDLSTH	(561; $231)	→	DLISTH	(54275; $D403)
SDLSTL	(560; $230)	→	DLISTL	(54274; $D402)
SDMCTL	(559; $22F)	→	DMACTL	(54272; $D400)
GPRIOR	(623; $26F)	→	PRIOR	(53275; $D01B)

Die folgende Routine ist wiederum nur auf XL/XE-Geräten vorhanden: Es wird überprüft, ob der Fine-Scrolling-Modus des Editors (→ Editor) eingeschaltet ist. Dazu wird VSFLAG (620; $26C) abgefragt. Enthält es einen Wert ungleich 0, dann wird die fällige Änderung des Registers für vertikales Feinverschieben vorgenommen und VSFLAG um 1 erniedrigt.

Durch das Schreiben von 8 in CONSOL (53279; $D01F) wird daraufhin der (auf XL/XE-Geräten über den normalen Tonausgang simulierte) Tastaturlautsprecher abgeschaltet. Es folgt die Übertragung der Farben aus den Schattenregistern (ab PCOLR0, 704; $2C0) in die Hardwareregister (ab COLPM0, 53266; $D012). Dabei wird im ATRACT-Modus auch die notwendige Veränderung der Farben vorgenommen (d.h., dass im ATRACT-Modus der Wert der Schatten-Farbregister unverändert bleibt). Außerdem werden CHBAS (756; $2F4) in CHBASE (54281; $D409) und CHACT (755; $2F3) in CHACTL (54273; $D401) übertragen.

Daraufhin wird der zweite System-Timer CDTMV2 (538,539; $21A,$21B) erniedrigt (wenn er nicht 0 war) und, falls dieser den Wert 0 erreicht hat, über den Vektor CDTMA2 (552,553; $228,$229) das dazugehörige Unterprogramm aufgerufen. Es folgt die Bearbeitung des dritten, vierten und fünften Timers. Jeder der drei Zähler (ab CDTMV3, 540,541; $21C,$21D) wird um 1 vermindert (wenn er nicht 0 war). Wenn der Zähler dabei den Wert 0 erreicht hat, wird das zum Timer gehörige Flag gesetzt (CDTMF3, 554; $22A / CDTMF4, 556; $22C / CDTMF5, 558; $22E).

Der folgende Teil der VBI-Routine sorgt für die automatische Tastenwiederholfunktion. Auch hier gibt es einige Unterschiede zwischen den "alten" und den XL/XE-Geräten: Bei den Letzteren wird nämlich berücksichtigt, dass die Tastatur über KEYDIS (621; $26D) abgeschaltet sein könnte. Außerdem ist von der Tastenwiederholfunktion die HELP-Taste ausgeschlossen. Schließlich wird die Tastenwiederholfrequenz nicht mehr über eine Konstante, sondern über die Variable KEYREP (730; $2DA) festgelegt. Zuletzt errechnet die VBI-Routine die Werte für die Schattenregister für Joysticks, Paddles und Feuerknöpfe. Auch hier gibt es eine Abweichung zwischen alten und neuen Geräten: Bei den XL/XE-Modellen werden alle Werte, die den dritten und vierten Anschluss betreffen, jeweils vom ersten und zweiten Anschluss übernommen.

Die interne VBI-Routine wird durch einen Sprung durch den Vektor VVBLKD (548,549; $224,$225) beendete. Dieser Vektor zeigt normalerweise auf die Routine XITVBV (58466; $E462), die alle Datenregister und Statusregister des unterbrochenen Programms wieder installiert und dorthin zurückspringt.

Wie kann man nun die VBI-Routine nach seinen eigenen Bedürfnissen verändern? Dazu muss man zunächst mehr über den Aufbau der VBI-Verarbeitung erfahren. Das Betriebssystem stellt für eigene VBI-Routinen zwei Vektoren zur Verfügung, und zwar VVBLKI (546,547; $222,$223) und VVBKLD (548,549; $224,$225). Der Erstere zeigt auf die VBI-Routine, die ungeachtet aller zeitkritischen Vorgänge ausgeführt werden soll (diese VBI-Routine wird "Immediate VBI" genannt). Dieser Programmteil darf eine bestimmte Länge auf keinen Fall überschreiten, da sonst sowohl Bildschirmaufbau als auch die Ein- und Ausgabevorgänge gestört werden könnten. Dieser Vektor zeigt normalerweise auf den Anfang der internen VBI-Routine. Der zweite Vektor zeigt auf den Teil des VBI, der nach der internen Routine ausgeführt werden soll. Dieser Teil des VBI wird "Deferred VBI" genannt und wird nur dann angesprungen, wenn keine zeitkritischen Vorgänge im Gange sind.

Um korrekt mit eigenen Interrupt-Routinen arbeiten zu können, muss man die drei folgenden Routinen im Betriebssystem kennen:

SETVBV $E45C 58460

X: High-Byte
Y: Low-Byte
A: 1-5 → Timer 1-5
6 → Immediate VBI
7 → Deferred VBI

Diese Routine dient dazu, die Werte für die internen Timer und die Vektoren für die VB-Routinen zu setzen. Natürlich wäre es auch denkbar, die Vektoren und Zähler direkt zu verändern, wenn man dafür Sorge trägt, dass nicht ausgerechnet in dem Moment, in dem erst eines der beiden Bytes verändert ist, auf dieses Byte zugegriffen wird. Im Falle der VB-Vektoren würde dies zu einem Sprung zu einer falschen Adresse und damit wahrscheinlich zu einem Absturz des Programms führen.

Dazu muss man das X- und Y-Register mit dem High und Low Byte des betreffenden Wertes laden. Dieser Wert ist für die Timer der Anfangswert des Zählers, für die VB-Routinen die Adresse der jeweiligen Routine. Der Akkumulator gibt an, welcher Vektor oder Wert verändert werden soll.

SYSVBV $E45F 58463

Diese Routine sollte am Ende der eigenen Immediate-VB-Routine aufgerufen werden. Sie ruft, unabhängig von der Betriebssystemversion, die interne VB-Routine auf.

XITVBV $E462 58466

Diese Routine sollte am Ende der eigenen Deferred-VB-Routine aufgerufen werden. Sie sorgt dafür, dass das unterbrochene Hauptprogramm in korrekter Weise fortgesetzt werden kann.

Aus den beiden letzten beschriebenen Routinen geht auch hervor, wie die interne VB-Routine ganz unterdrückt werden kann. In diesem Fall ist es einfacher, eine einzige Routine zu schreiben, die als Immediate-VB aufgerufen und wie ein Deferred-VB verlassen wird. Zwei Dinge sollten dabei beachtet werden: Alles, was normalerweise die interne VB-Routine macht und für das eigene Programm von Bedeutung ist, muss die eigene VB-Routine erledigen; finden im eigenen Programm zeitkritische Vorgänge statt, dann sollte man selbst seine VB-Routine in einen "Immediate"-Teil und einen "Deferred"-Teil aufteilen (siehe dazu die Beschreibung der internen VB-Routine).

3 Tabellenteil

Erläuterung

Die im ursprünglichen Profibuch enthaltene große Vergleichstabelle zum ATARI 8-Bit-Computer wurde zwecks einfacherer Lesbarkeit in verschiedene Tabellen aufgeteilt. Diese bieten eine vollständige Übersicht über die Codes 0-255 und ihre jeweilige Funktion bzw. Bedeutung. Angegeben sind ihre Werte in dezimaler, hexadezimaler und binärer Form, ergänzt um die dazugehörigen Zeichen in ASCII, ATASCII, internem Code und Tastencode.

Die Tastencodes wurden ergänzt um die sogenannten F-Tasten F1-F4, die seit dem ATARI 1200XL im OS des ATARI XL/XE vorhanden und mit bestimmten Funktionen belegt sind. Daneben lassen sich 6 weitere F-Tasten nutzen, wenn man diese zusätzlich einbaut. Sie können abgefragt und frei benutzt werden, da sie vom OS nicht belegt sind. Die Informationen dazu sind in rechteckige Klammern [F5] bis [F10] gesetzt.

Die Mnemonics wurden in separate Tabellen ausgelagert und um wichtige Details für Programmierer ergänzt.

Der Tabellenteil besteht nun aus:

Allgemeine Übersicht der Systemcodes

Übersicht der Operation Codes der CPU

Alphabetische Liste der Systemadressen

Systemcodes

Dez	Hex	Binär	ASCII	ATASCII		Intern	Tastenkombi	Tastencode von
0	$00	%00000000	NULL	♥	á		CTRL-,	L
1	$01	%00000001	SOH	├	ù	!	CTRL-A	J
2	$02	%00000010	STX	▕	Ñ	"	CTRL-B	;
3	$03	%00000011	ETX	┘	É	#	CTRL-C	F1 (X)
4	$04	%00000100	EOT	┤	ç	$	CTRL-D	F2 (X)
5	$05	%00000101	END	┐	ô	%	CTRL-E	K
6	$06	%00000110	ACK	╱	ò	&	CTRL-F	+
7	$07	%00000111	BEL	╲	ì	'	CTRL-G	*
8	$08	%00001000	BS	◢	£	(	CTRL-H	O
9	$09	%00001001	HT	▗	ï	)	CTRL-I	[F7] (X)
10	$0A	%00001010	LF	◣	ü	*	CTRL-J	P
11	$0B	%00001011	VT	▝	ä	+	CTRL-K	U
12	$0C	%00001100	FF	▘	Ö	,	CTRL-L	Return
13	$0D	%00001101	CR	▔	ú	-	CTRL-M	I
14	$0E	%00001110	SO	▁	ó	.	CTRL-N	-
15	$0F	%00001111	SI	▖	ö	/	CTRL-O	=
16	$10	%00010000	DLE	♣	Ü	0	CTRL-P	V
17	$11	%00010001	DC1	┌	â	1	CTRL-Q	Help
18	$12	%00010010	DC2	─	û	2	CTRL-R	C
19	$13	%00010011	DC3	┼	î	3	CTRL-S	F3 (X)
20	$14	%00010100	DC4	●	é	4	CTRL-T	F4 (X)
21	$15	%00010101	NAK	▄	è	5	CTRL-U	B
22	$16	%00010110	SYN	▎	ñ	6	CTRL-V	X
23	$17	%00010111	ETB	┬	ê	7	CTRL-W	Z
24	$18	%00011000	CAN	┴	å	8	CTRL-X	4
25	$19	%00011001	EM	▌	à	9	CTRL-Y	[F6] (X)
26	$1A	%00011010	SUB	└	Å	:	CTRL-Z	3
27	$1B	%00011011	ESC	␛		␛	ESC/ESC	6
28	$1C	%00011100	FS	↑		↑	ESC-CTRL- -	ESC
29	$1D	%00011101	GS	↓		↓	ESC-CTRL- =	5
30	$1E	%00011110	RS	←		←	ESC-CTRL- +	2
31	$1F	%00011111	US	→		→	ESC-CTRL- *	1

ESC → ESC-Taste 1x drücken / INV → Invers-Taste

Systemcodes

Dez	Hex	Binär	ASCII	ATASCII	Intern	Tastenkombi	Tastencode von
32	$20	%00100000			@	SPACE	,
33	$21	%00100001	!	!	A	SHIFT-1	SPACE
34	$22	%00100010	"	"	B	SHIFT-2	.
35	$23	%00100011	#	#	C	SHIFT-3	N
36	$24	%00100100	$	$	D	SHIFT-4	[F9] (X)
37	$25	%00100101	%	%	E	SHIFT-5	M
38	$26	%00100110	&	&	F	SHIFT-6	/
39	$27	%00100111	'	'	G	SHIFT-7	INV
40	$28	%00101000	(	(	H	SHIFT-9	R
41	$29	%00101001	)	)	I	SHIFT-0	[F8] (X)
42	$2A	%00101010	*	*	J	*	E
43	$2B	%00101011	+	+	K	+	Y
44	$2C	%00101100	,	,	L	,	TAB
45	$2D	%00101101	-	-	M	-	T
46	$2E	%00101110	.	.	N	.	W
47	$2F	%00101111	/	/	O	/	Q
48	$30	%00110000	0	0	P	0	9
49	$31	%00110001	1	1	Q	1	[F5] (X)
50	$32	%00110010	2	2	R	2	0
51	$33	%00110011	3	3	S	3	7
52	$34	%00110100	4	4	T	4	BCKSP
53	$35	%00110101	5	5	U	5	8
54	$36	%00110110	6	6	V	6	<
55	$37	%00110111	7	7	W	7	>
56	$38	%00111000	8	8	X	8	F
57	$39	%00111001	9	9	Y	9	H
58	$3A	%00111010	:	:	Z	SHIFT-;	D
59	$3B	%00111011	;	;	[	;	[F10] (X)
60	$3C	%00111100	<	<	\	<	CAPS
61	$3D	%00111101	=	=	]	=	G
62	$3E	%00111110	>	>	^	>	S
63	$3F	%00111111	?	?	_	SHIFT-/	A

BCKSP → 'BACK SPACE'-Taste / SPACE → Leertaste

Systemcodes

Dez	Hex	Binär	ASCII		ATASCII	Intern		Tastenkombi	Tastencode von
64	$40	%01000000	@		@	♥	á	á	SHIFT-B
65	$41	%01000001	A		A	├	ù	ù	A
66	$42	%01000010	B		B	▕	Ñ	Ñ	B
67	$43	%01000011	C		C	┘	É	É	C
68	$44	%01000100	D		D	┤	ç	ç	D
69	$45	%01000101	E		E	┐	ô	ô	E
70	$46	%01000110	F		F	╱	ò	ò	F
71	$47	%01000111	G		G	╲	ì	ì	G
72	$48	%01001000	H		H	◢	£	£	H
73	$49	%01001001	I		I	▗	ï	ï	I
74	$4A	%01001010	J		J	◣	ü	ü	J
75	$4B	%01001011	K		K	▝	ä	ä	K
76	$4C	%01001100	L		L	▘	Ö	Ö	L
77	$4D	%01001101	M		M	▔	ú	ú	M
78	$4E	%01001110	N		N	▁	ó	ó	N
79	$4F	%01001111	O		O	▖	ö	ö	O
80	$50	%01010000	P		P	♣	Ü	Ü	P
81	$51	%01010001	Q		Q	┌	â	â	Q
82	$52	%01010010	R		R	─	û	û	R
83	$53	%01010011	S		S	┼	î	î	S
84	$54	%01010100	T		T	●	é	é	T
85	$55	%01010101	U		U	▄	è	è	U
86	$56	%01010110	V		V	▎	ñ	ñ	V
87	$57	%01010111	W		W	┬	ê	ê	W
88	$58	%01011000	X		X	┴	å	å	X
89	$59	%01011001	Y		Y	▌	à	à	Y
90	$5A	%01011010	Z		Z	└	Å	Å	Z
91	$5B	%01011011	[	Ä	[	␛		SHIFT- ,	SHIFT-6
92	$5C	%01011100	\	Ö	\	↑		SHIFT- +	SHIFT-ESC
93	$5D	%01011101	]	Ü	]	↓		SHIFT- .	SHIFT-5
94	$5E	%01011110	^		^	←		SHIFT- *	SHIFT-2
95	$5F	%01011111	_		_	→		SHIFT- –	SHIFT-1

ESC → ESC-Taste 1x drücken / INV → Invers-Taste

Systemcodes

Dez	Hex	Binär	ASCII	ATASCII	Intern	Tastenkombi	Tastencode von
96	$60	%01100000	'	♦ i	♦ i	CTRL- .	SHIFT- ;
97	$61	%01100001	a	a	a	A	SHIFT-BCKSP
98	$62	%01100010	b	b	b	B	SHIFT- .
99	$63	%01100011	c	c	c	C	SHIFT-N
100	$64	%01100100	d	d	d	D	[SHIFT-F9] (X)
101	$65	%01100101	e	e	e	E	SHIFT-M
102	$66	%01100110	f	f	f	F	SHIFT- /
103	$67	%01100111	g	g	g	G	SHIFT-INV
104	$68	%01101000	h	h	h	H	SHIFT-R
105	$69	%01101001	i	i	i	I	[SHIFT-F8] (X)
106	$6A	%01101010	j	j	j	J	SHIFT-E
107	$6B	%01101011	k	k	k	K	SHIFT-Y
108	$6C	%01101100	l	l	l	L	SHIFT-TAB
109	$6D	%01101101	m	m	m	M	SHIFT-T
110	$6E	%01101110	n	n	n	N	SHIFT-W
111	$6F	%01101111	o	o	o	O	SHIFT-Q
112	$70	%01110000	p	p	p	P	SHIFT-9
113	$71	%01110001	q	q	q	Q	[SHIFT-F5] (X)
114	$72	%01110010	r	r	r	R	SHIFT-0
115	$73	%01110011	s	s	s	S	SHIFT-7
116	$74	%01110100	t	t	t	T	SHIFT-BCKSP
117	$75	%01110101	u	u	u	U	SHIFT-8
118	$76	%01110110	v	v	v	V	SHIFT-<
119	$77	%01110111	w	w	w	W	SHIFT->
120	$78	%01111000	x	x	x	X	SHIFT-F
121	$79	%01111001	y	y	y	Y	SHIFT-H
122	$7A	%01111010	z	z	z	Z	SHIFT-D
123	$7B	%01111011	{ ä	♠	♠	CTRL- ;	[SHIFT-F10] (X)
124	$7C	%01111100	| ö	▌	▌	SHIFT- =	SHIFT-CAPS
125	$7D	%01111101	} ü	↖	↖	ESC/SHIFT-<	SHIFT-G
126	$7E	%01111110	~ ß	◀	◀	ESC/BCKSP	SHIFT-S
127	$7F	%01111111	DEL	▶	▶	ESC/TAB	SHIFT-A

BCKSP → 'BACK SPACE'-Taste / SPACE → Leertaste

Systemcodes

Dez	Hex	Binär	ASCII	ATASCII	Intern	Tastenkombi	Tastencode von
128	$80	%10000000				INV / CTRL-,	CTRL-L
129	$81	%10000001				INV / CTRL-A	CTRL-J
130	$82	%10000010				INV / CTRL-B	CTRL- ;
131	$83	%10000011				INV / CTRL-C	CTRL-F1 (X)
132	$84	%10000100				INV / CTRL-D	CTRL-F2 (X)
133	$85	%10000101				INV / CTRL-E	CTRL-K
134	$86	%10000110				INV / CTRL-F	CTRL- +
135	$87	%10000111				INV / CTRL-G	CTRL- *
136	$88	%10001000				INV / CTRL-H	CTRL-O
137	$89	%10001001				INV / CTRL-I	[CTRL-F7] (X)
138	$8A	%10001010				INV / CTRL-J	CTRL-P
139	$8B	%10001011				INV / CTRL-K	CTRL-U
140	$8C	%10001100				INV / CTRL-L	CTRL-RETURN
141	$8D	%10001101				INV / CTRL-M	CTRL-I
142	$8E	%10001110				INV / CTRL-N	CTRL- –
143	$8F	%10001111				INV / CTRL-O	CTRL- =
144	$90	%10010000				INV / CTRL-P	CTRL-V
145	$91	%10010001				INV / CTRL-Q	CTRL-HELP
146	$92	%10010010				INV / CTRL-R	CTRL-C
147	$93	%10010011				INV / CTRL-S	CTRL-F3 (X)
148	$94	%10010100				INV / CTRL-T	CTRL-F4 (X)
149	$95	%10010101				INV / CTRL-U	CTRL-B
150	$96	%10010110				INV / CTRL-V	CTRL-X
151	$97	%10010111				INV / CTRL-W	CTRL-Z
152	$98	%10011000				INV / CTRL-X	CTRL-4
153	$99	%10011001				INV / CTRL-Y	[CTRL-F6] (X)
154	$9A	%10011010				INV / CTRL-Z	CTRL-3
155	$9B	%10011011				RETURN	CTRL-6
156	$9C	%10011100				ESC-SHIFT-BCKSP	CTRL-ESC
157	$9D	%10011101				ESC-SHIFT->	CTRL-5
158	$9E	%10011110				ESC-CTRL-TAB	CTRL-2
159	$9F	%10011111				ESC-SHIFT-TAB	CTRL-1

ESC → ESC-Taste 1x drücken / INV → Invers-Taste

Systemcodes

Dez	Hex	Binär	ASCII	ATASCII	Intern	Tastenkombi	Tastencode von
160	$A0	%10100000		█	@	INV / SPACE	CTRL- ,
161	$A1	%10100001		!	A	INV / SHIFT-1	CTRL-SPACE
162	$A2	%10100010		"	B	INV / SHIFT-2	CTRL- .
163	$A3	%10100011		#	C	INV / SHIFT-3	CTRL-N
164	$A4	%10100100		$	D	INV / SHIFT-4	[CTRL-F9] (X)
165	$A5	%10100101		%	E	INV / SHIFT-5	CTRL-M
166	$A6	%10100110		&	F	INV / SHIFT-6	CTRL- /
167	$A7	%10100111		'	G	INV / SHIFT-7	CTRL-INV
168	$A8	%10101000		(	H	INV / SHIFT-9	CTRL-R
169	$A9	%10101001		)	I	INV / SHIFT-0	[CTRL-F8] (X)
170	$AA	%10101010		*	J	INV / *	CTRL-E
171	$AB	%10101011		+	K	INV / +	CTRL-Y
172	$AC	%10101100		,	L	INV / ,	CTRL-TAB
173	$AD	%10101101		-	M	INV / –	CTRL-T
174	$AE	%10101110		.	N	INV / .	CTRL-W
175	$AF	%10101111		/	O	INV / /	CTRL-Q
176	$B0	%10110000		0	P	INV / 0	CTRL-9
177	$B1	%10110001		1	Q	INV / 1	[CTRL-F5] (X)
178	$B2	%10110010		2	R	INV / 2	CTRL-0
179	$B3	%10110011		3	S	INV / 3	CTRL-7
180	$B4	%10110100		4	T	INV / 4	CTRL-BCKSP
181	$B5	%10110101		5	U	INV / 5	CTRL-8
182	$B6	%10110110		6	V	INV / 6	CTRL-<
183	$B7	%10110111		7	W	INV / 7	CTRL->
184	$B8	%10111000		8	X	INV / 8	CTRL-F
185	$B9	%10111001		9	Y	INV / 9	CTRL-H
186	$BA	%10111010		:	Z	INV / SHIFT- ;	CTRL-D
187	$BB	%10111011		;	[	INV / ;	[CTRL-F10] (X)
188	$BC	%10111100		<	\	INV / <	CTRL-CAPS
189	$BD	%10111101		=	]	INV / =	CTRL-G
190	$BE	%10111110		>	^	INV / >	CTRL-S
191	$BF	%10111111		?	_	INV / SHIFT- /	CTRL-A

BCKSP → 'BACK SPACE'-Taste / SPACE → Leertaste

Systemcodes

Dez	Hex	Binär	ASCII	ATASCII	Intern		Tastenkombi	Tastencode von
192	$C0	%11000000					INV / SHIFT-8	
193	$C1	%11000001					INV / A	
194	$C2	%11000010					INV / B	
195	$C3	%11000011					INV / C	SHIFT-CTRL-F1 (X)
196	$C4	%11000100					INV / D	SHIFT-CTRL-F2 (X)
197	$C5	%11000101					INV / E	
198	$C6	%11000110					INV / F	
199	$C7	%11000111					INV / G	
200	$C8	%11001000					INV / H	SHIFT-CTRL-O
201	$C9	%11001001					INV / I	[SHIFT-CTRL-F7] (X)
202	$CA	%11001010					INV / J	SHIFT-CTRL-P
203	$CB	%11001011					INV / K	SHIFT-CTRL-U
204	$CC	%11001100					INV / L	SHIFT-CTRL-RETURN
205	$CD	%11001101					INV / M	SHIFT-CTRL-I
206	$CE	%11001110					INV / N	SHIFT-CTRL- –
207	$CF	%11001111					INV / O	SHIFT-CTRL- =
208	$D0	%11010000					INV / P	
209	$D1	%11010001					INV / Q	
210	$D2	%11010010					INV / R	
211	$D3	%11010011					INV / S	SHIFT-CTRL-F3 (X)
212	$D4	%11010100					INV / T	SHIFT-CTRL-F4 (X)
213	$D5	%11010101					INV / U	
214	$D6	%11010110					INV / V	
215	$D7	%11010111					INV / W	
216	$D8	%11011000					INV / X	SHIFT-CTRL-4
217	$D9	%11011001					INV / Y	[SHIFT-CTRL-F6] (X)
218	$DA	%11011010					INV / Z	SHIFT-CTRL-3
219	$DB	%11011011					INV / SHIFT- ,	SHIFT-CTRL-6
220	$DC	%11011100					INV / SHIFT- +	SHIFT-CTRL-ESC
221	$DD	%11011101					INV / SHIFT- .	SHIFT-CTRL-5
222	$DE	%11011110					INV / SHIFT- *	SHIFT-CTRL-2
223	$DF	%11011111					INV / SHIFT- –	SHIFT-CTRL-1

ESC → ESC-Taste 1x drücken / INV → Invers-Taste

Systemcodes

Dez	Hex	Binär	ASCII	ATASCII	Intern	Tastenkombi	Tastencode von
224	$E0	%11100000		i	i	INV / CTRL- .	SHIFT-CTRL- ,
225	$E1	%11100001				INV / A	SHIFT-CTRL-BCKSP
226	$E2	%11100010				INV / B	SHIFT-CTRL- .
227	$E3	%11100011				INV / C	SHIFT-CTRL-N
228	$E4	%11100100				INV / D	[SHIFT-CTRL-F9] (X)
229	$E5	%11100101				INV / E	SHIFT-CTRL-M
230	$E6	%11100110				INV / F	SHIFT-CTRL- /
231	$E7	%11100111				INV / G	SHIFT-CTRL-INV
232	$E8	%11101000				INV / H	SHIFT-CTRL-R
233	$E9	%11101001				INV / I	[SHIFT-CTRL-F8] (X)
234	$EA	%11101010				INV / J	SHIFT-CTRL-E
235	$EB	%11101011				INV / K	SHIFT-CTRL-Y
236	$EC	%11101100				INV / L	SHIFT-CTRL-TAB
237	$ED	%11101101				INV / M	SHIFT-CTRL-T
238	$EE	%11101110				INV / N	SHIFT-CTRL-W
239	$EF	%11101111				INV / O	SHIFT-CTRL-Q
240	$F0	%11110000				INV / P	SHIFT-CTRL-9
241	$F1	%11110001				INV / Q	[SHIFT-CTRL-F5] (X)
242	$F2	%11110010				INV / R	SHIFT-CTRL-0
243	$F3	%11110011				INV / S	SHIFT-CTRL-7
244	$F4	%11110100				INV / T	SHIFT-CTRL-BCKSP
245	$F5	%11110101				INV / U	SHIFT-CTRL-8
246	$F6	%11110110				INV / V	SHIFT-CTRL- <
247	$F7	%11110111				INV / W	SHIFT-CTRL- >
248	$F8	%11111000				INV / X	SHIFT-CTRL-F
249	$F9	%11111001				INV / Y	SHIFT-CTRL-H
250	$FA	%11111010				INV / Z	SHIFT-CTRL-D
251	$FB	%11111011				INV / CTRL- ;	[SHIFT-CTRL-F10] (X)
252	$FC	%11111100				INV / SHIFT- =	SHIFT-CTRL-CAPS
253	$FD	%11111101				ESC / CTRL-2	SHIFT-CTRL-G
254	$FE	%11111110				ESC / CTRL-BCKSP	SHIFT-CTRL-S
255	$FF	%11111111				ESC / CTRL- >	SHIFT-CTRL-A

BCKSP → 'BACK SPACE'-Taste / SPACE → Leertaste

Übersicht 6502 Operation Codes (Opcodes)

Der NMOS 6502-Prozessor erzeugt mit 56 Mnemonics 151 Opcodes.

	x0		x1		x2		x3		x4		x5		x6		x7	
0x	BRK		ORA(z,X)								ORA z		ASL z			
	1	7	2	6							2	2	2	5		
1x	BPL r		ORA(z),Y								ORA z,X		ASL z,X			
	2	2/3+	2	5+							2	3	2	6		
2x	JSR a		AND(z,X)						BIT z		AND z		ROL z			
	3	6	2	6					2	3	2	2	2	5		
3x	BMI r		AND(z),Y								AND z,X		ROL z,X			
	2	2/3+	2	5+							2	3	2	6		
4x	RTI		EOR(z,X)								EOR z		LSR z			
	1	6	2	6							2	3	2	5		
5x	BVC r		ADC(z),Y								EOR z,X		LSR z,X			
	2	2/3+	2	5+							2	4	2	6		
6x	RTS		ADC(z,X)								ADC z		ROR z			
	1	6	2	6							2	3	2	5		
7x	BVS r		ADC(z),Y								ADC z,X		ROR z,X			
	2	2/3+	2	5+							2	4	2	6		
8x			STA(z,X)						STY z		STA z		STX z			
			2	6					2	3	2	3	2	3		
9x	BCC r		STA(z),Y						STY z,X		STA z,X		STX z,X			
	2	2/3+	2	6					2	4	2	4	2	4		
Ax	LDY #		LDA(z,X)		LDX #				LDY z		LDA z		LDX z			
	2	2	2	6	2	2			2	3	2	3	2	3		
Bx	BCS r		LDA(z),Y						LDY z,X		LDA z,X		LDX z,X			
	2	2/3+	2	5+					2	4	2	4	2	4		
Cx	CPY #		CMP(z,X)						CPY z		CMP z		DEC z			
	2	2	2	6					2	3	2	3	2	5		
Dx	BNE r		CMP(z),Y								CMP z,X		DEC z,X			
	2	2/3+	2	5+							2	4	2	6		
Ex	CPX #		SBC(z,X)						CPX z		SBC z		INC z			
	2	2	2	6					2	3	2	3	2	5		
Fx	BEQ r		SBC(z),Y								SBC z,X		INC z,X			
	2	2/3+	2	5+							2	4				

Hinweis: + = plus 1 CPU-Zyklus bei überschreiten einer Page-Grenze

<u>Erläuterung:</u>

obere Zeile enthält den Opcode: <u>PHP</u>
darunter Anzahl an Bytes | Anzahl CPU-Zyklen: 1|3

	x8		x9		xA		xB		xC		xD		xE		xF	
0x	PHP		ORA #		ASL A						ORA a		ASL a			
	1	3	2	2	1	2					3	4	3	6		
1x	CLC		ORA a,Y								ORA a,X		ASL a,X			
	1	2	3	4+							3	4+	3	7		
2x	PLP		AND #		ROL A				BIT a		AND a		ROL a			
	1	4	2	2	1	2			3	4	3	4	3	6		
3x	SEC		AND a,Y								AND a,X		ROL a,X			
	1	2	3	4+							3	4+	3	7		
4x	PHA		EOR #		LSR A				JMP a		EOR a		LSR a			
	1	3	2	2	1	2			3	3	3	4	3	6		
5x	CLI		EOR a,Y								EOR a,X		LSR a,X			
	1	2	3	4+							3	4+	3	7		
6x	PLA		ADC #		ROR A				JMP(a)		ADC a		ROR a			
	1	4	2	2	1	2			3	5	3	4	3	6		
7x	SEI		ADC a,Y								ADC a,X		ROR a,X			
	1	2	3	4+							3	4+	3	7		
8x	DEY				TXA				STY a		STA a		STX a			
	1	2			1	2			3	4	3	4	3	4		
9x	TYA		STA a,Y		TXS						STA a,X					
	1	2	3	5	1	2					3	5				
Ax	TAY		LDA #		TAX				LDY a		LDA a		LDX a			
	1	2	2	2	1	2			3	4	3	4	3	4		
Bx	CLV		LDA a,Y		TSX				LDY a,X		LDA a,X		LDX a,Y			
	1	2	3	4+	1	2			3	4+	3	4+	3	4+		
Cx	INY		CMP #		DEX				CPY a		CMP a		DEC a			
	1	2	2	2	1	2			3	4	3	4	3	6		
Dx	CLD		CMP a,Y								CMP a,X		DEC a,X			
	1	2	3	4+							3	4+	3	7		
Ex	INX		SBC #		NOP				CPX a		SBC a		INC a			
	1	2	2	2	1	2			3	4	3	4	3	6		
Fx	SED		SBC a,Y								SBC a,X		INC a,X			
	1	2	3	4+							3	4+	3	7		

Branch: 2/3 = 2 Zyklen, falls nicht genommen, 3 Zyklen falls genommen

Alphabetische Liste der Systemadressen

28-31	$1C-$1F	ABUFPT (X)
727,728	$2D7,$208	ACMISR (X)
1005-1015	$360,$3F7	ACMUAR (X)
7B2	$30E	ADDCOR (A)
100,101	$64,$E5	ADRESS
55296	$D800	AFP
53768	$0208	ALLPOT
14,15	$E,$F	APPMHI
763	$2F8	ATACHR
77	$4D	ATRACT
53761	$0201	AUDC1
53763	$0203	AUDC2
53765	$D205	AUDC3
53767	$0207	AUDC4
53768	$0208	AUDCTL
53760	$D200	AUDF1
53762	$0202	AUDF2
53764	$0204	AUDF3
53766	$D206	AUDF4
1016	$3F8	BASICF (X)
52,53	$34,$35	BFENLO/HI
110	$6E	BITMSK
212,213	$D4,$D5	BININT
650	$28A	BLIM
58481	$E471	BLKBDV
9	$9	BOOT?
578,579	$242,$243	BOOTAD
703	$2BF	BOTSCR
61	$3D	BETR
17	$11	BRKKEY
566,567	$236,$237	BRKKY (X)
21,22	$15,$16	BUFADR
107	$68	BUFCNT
56	$38	BUFRFL

50,51	$32,$33	BUFRLO/HI
108,109	$6C,$60	BUFSTR
49150,49151	$BFFE,$BFFF	CARTAD
1003	$3E8	CARTCK (X)
49146,49147	$BFFA,$BFFB	CARTCS
49149	$BFFD	CARTFG
49148	$BFFC	CART
1021-1151	$3FD-$47F	CASBUF
58432	$E440	CASETV
783	$30F	CASFLG
2,3	$2,$3	CASINI
75	$48	CASSBT (A)
1002	$3EA	CASSBT (X)
572,573	$23C,$23D	CAUX1,2
750,751	$2EE,$2EF	CBAUDL/H
571	$238	CCOMND
570	$23A	CDEVIC
550,551	$226,$227	CDTMA1
552,553	$228,$229	CDTMA2
554	$224	CDTMF3
556	$22C	CDTMF4
558	$22E	CDTMF5
536,537	$218,$219	CDTMV1
538,539	$21A,$218	CDTMV2
540,541	$21C,$21D	CDTMV3
542,543	$21E,$21F	CDTMV4
544,545	$220,$221	CDTMV5
754	$2F2	CH1
54273	$D401	CHACTL
755	$2F3	CHACT
762	$2FA	CHAR
54281	$D409	CHBASE
756	$2F4	CHBAS
59	$38	CHKSNT
49	$31	CHKSUM
1019,1020	$3FB,$3FC	CHLINK (X)

240	$F0	CHRFLG
619	$268	CHSALT (X)
764	$2FC	CH
47	$2F	CIOCHR
58478	$E46E	CIOINV
58454	$E456	CIOV
242	$F2	CIX
74	$4A	CKEY (A)
1001	$3E9	CKEY (X)
114,115	$72,$73	COLAC
53274	$001A	COLBK
85,86	$55,556	COLCRS
580	$244	COLDST
58487	$E477	COLDSV
122	$7A	COLINC (A)
761	$2F9	COLINC (X)
708	$2C4	COLOR0
709	$265	COLOR1
710	$2C6	COLOR2
711	$2C7	COLOR3
712	$2C8	COLOR4
53270	$D016	COLPF0
53271	$D017	COLPF1
53272	$D018	COLPF2
53273	$D019	COLPF3
53266	$D012	COLPM0
53267	$D013	COLPM1
53268	$D014	COLPM2
53269	$D015	COLPM3
79	$4F	COLRSH
53279	$D01F	CONSOL
126,127	$7E,$7F	COUNTR
54	$36	CRETRY (A)
668	$29C	CRETRY (X)
66	$42	CRITIC
752	$2F0	CRSINH

58493	$E47D	CSOPIV
648	$288	CSTAT (A)
778,779	$30A,$308	DAUX1/2
577	$241	DBSECT
772,773	$304,$305	DBUFLO/HI
776,777	$308,$309	DBYTLO/HI
770	$302	DCOMND
768	$300	DDEVIC
1004	$3EC	DEERF (X)
119,120	$77,$78	DELTAC
118	$76	DELTAR
576	$240	DFLAGS
241	$F1	DIGRT
87	$57	DINDEX
5301	$1485	DIOV (X)
54274,54275	$D402,$D403	DLISTL/H
54272	$D400	DMACTL
733	$2DD	DMASAV (X)
672	$2A0	DMASK
12,13	$C,$D	DOSINI
10,11	$A,$B	DOSVEC
55	$37	DRETRY (A)
701	$28D	DRETRY (X)
78	$4E	DRKMSK
725,726	$2D5,$2D6	DSCTLN (X)
24,25	$18,$19	DSKFMS
58451	$E453	DSKINV
58488	$E478	DSKIV
582	$246	DSKTIM
26,27	$1A,$18	DSKUTL
766	$2FE	DSPFLG
771	$303	DSTATS
76	$4C	DSTAT
774	$306	DTIMLO
5300	$14B4	DTYPE (X)
769	$301	DUNIT

775	$307	DUNUSE
746-749	$2EA-$2ED	DVSTAT
58368	$E400	EDITRV
237	$ED	EEXP
116,117	$74,$75	ENDFT
575	$23F	ERRFLG
73	$49	ERRNO
195	$C3	ERRSAV
674	$2A2	ESCFLG
239	$EF	ESIGN
56780	$DDCC	EXP10
56768	$DDC0	EXP
55910	$DA66	FADD
55526	$D8E6	FASC
56104	$D828	FDIV
63	$3F	FEOF
765	$2FD	FILDAT
695	$287	FILFLG
622	$26E	FINE (X)
96,97	$60,$61	FKDEF (X)
56717	$DD8D	FLD0P
56713	$DD89	FLD0R
56732	$DD9C	FLD1P
56728	$DD98	FLD1R
252,253	$FC,$FD	FLPTR
56758	$DDB6	FMOVE
56027	$DADB	FMUL
67-73	$43-$49	FMZSPG
55762	$D9D2	FPI
254,255	$FE,$FF	FPTR2
212-217	$D4-$D9	FR0
224-229	$60-$E5	FR1
230-235	$E6-$EB	FR2
64	$40	FREQ
218-223	$DA-$DF	FRE
236	$EC	FRX

56747	$DDAB	FST0P
56743	$DDA7	FST0R
55904	$DA60	FSUB
62	$3E	FTYPE
719,720	$2CF,$2D0	GBYTEA (X)
1018	$3FA	GINTLK (X)
58511	$648F	GPDVV (X)
623	$26F	GPRIOR
53277	$D01D	GRACTL
53265	$D011	GRAFM
53261	$D00D	GRAFP0
53262	$D006	GRAFP1
53263	$D00F	GRAFP2
53264	$D010	GRAFP3
794-831	$31A-$33F	HATABS (A)
794-828	$31A-$33C	HATABS (X)
732	$2DC	HELPFG (X)
648	$280	HIBZTE (X)
53278	$D01E	HITCLR
715,716	$2CB,$2CC	HIUSED (X)
745	$2E9	HNDLOD (X)
81	$51	HOLD1
671	$29F	HOLD2
669	$29D	HOLD3
700	$2BC	HOLD4
701	$2BD	HOLD5 (A)
124	$7C	HOLDCH
53252	$D004	HPOSM0
53253	$D005	HPOSM1
53254	$D006	HPOSM2
53255	$D007	HPOSM3
53248	$D000	HPOSP0
53249	$D001	HPOSP1
53250	$D002	HPOSP2
53251	$D003	HPOSP3
54276	$D404	HSCROL

842	$34A	ICAX1
42	$2A	ICAX1Z
843	$348	ICAX2
43	$28	ICAX2Z
844	$34C	ICAX3
845	$34D	ICAX4
846	$34E	ICAX5
847	$34F	ICAX6
836,837	$344,$345	ICBADR
37	$25	ICBAHZ
36	$24	ICBALZ
840,841	$348,$349	ICBLEN
41	$29	ICBLHZ
40	$28	ICELLZ
23	$17	ICCOMT
834	$342	ICCOM
34	$22	ICCOMZ
833	$341	ICDNO
33	$21	ICDNOZ
832	$340	ICHID
32	$20	ICHIDZ
46	$2E	ICIDNO
39	$27	ICPTHZ
38	$26	ICPTLZ
838,839	$346,$347	ICPUT
44,45	$2C,$2D	ICSPRZ
835	$843	ICSTA
35	$23	ICSTAZ
55722	$D9AA	IFP
651	$288	IMASK (X)
243,244	$F3,$F4	INBUFF
738,739	$2E2,$2E3	INITAD
125	$7D	INSDAT
557	$22D	INTEMP
58475	$E468	INTINV
694	$286	INVFLG

53774	$D20E	IRDEN
53774	$D20E	IRQST
782	$30E	JMPERS (X)
652,653	$28C,$28D	JVECK (X)
53769	$D209	KBCODE
58400	$E420	KEYBDV
121,122	$79,$7A	KEYDEF (X)
753	$2F1	KEYDEL
621	$26D	KEYDIS (X)
730	$2DA	KEYREP (X)
729	$2D9	KRPDEL (X)
1406,1407	$57E,$57F	LBPR1,2
1408-1535	$580-$5FF	LBUFF
563	$233	LCOUNT (X)
583-622	$247-$26E	LINBUF (A)
0	$0	LINFLG (X)
0,1	$0,$1	LINZBS (A)
82	$52	LMARGN
721,722	$2D1,$2D2	LOADAD (X)
202	$CA	LOADFLG
57041	$DED1	LOG10
99	$63	LOGCOL
690-693	$282-$285	LOGMAP
57037	$DECD	LOG
128,129	$80,$81	LOMEM
564	$234	LPENH
565	$235	LPENV
54,55	$36,837	LTEMP (X)
53248	$D000	M0PF
53256	$0008	M0PL
53249	$0001	M1PF
53257	$0009	M1PL
53250	$D002	M2PF
53258	$D00A	M2PL
53251	$0003	M3PF
53259	$0008	M3PL

743,744	$2E7,$2E8	MEMLO
144,145	$90,$91	MEMTOP
741,742	$2E5,$2E6	MEMTOP
1017	$3F9	MINTLK (X)
102,103	$66,$67	MLTTMP
654,655	$28E,$28F	NEWADR (X)
97,98	$61,$62	NEWCOL (A)
758,759	$2F6,$2F7	NEWCOL (X)
96	$60	NEWROW (A)
757	$2F5	NEWROW (X)
1	$1	NGFLAG (X)
54286	$D406	NMIEN
54287	$D40F	NMIRES
54287	$D40F	NMIST
60	$3C	NOCKSM
731	$208	NOCLIK (X)
238	SEE	NSIGN
94,95	$5E,$5F	OLDADR
93	$50	OLDCHR
91,92	$58,$5C	OLDCOL
90	$5A	OLDROW
53252	$D004	P0PF
53260	$D00C	P0PL
53253	$D005	P1PF
53261	$D00D	P1PL
53254	$D006	P2PF
53262	$D00E	P2PL
53255	$D007	P3PF
53263	$D00F	P3PL
54018	$D302	PACTL
624	$270	PADDL0
625	$271	PADDL1
626	$272	PADDL2
627	$273	PADDL3
628	$274	PADDL4
629	$275	PADDL5

630	$276	PADDL6
631	$277	PADDL7
98	$62	PALNTS (X)
53268	$D014	PAL
54019	$D303	PBCTL
29	$1D	PBPNT (A)
734	$2DE	PBPNT (X)
30	$1E	PBUFSZ (A)
735	$2DF	PBUFSZ (X)
704	$2C0	PCOLR0
705	$2C1	PCOLR1
706	$2C2	PCOLR2
707	$2C3	PCOLR2
55296,55297	$D800,$D801	PDCKSM (X)
55299	$D803	PDID1 (X)
55307	$D808	PDID2 (X)
585	$249	PDIMSK (X)
55301	$D805	PDIOV (X)
55304	$D808	PDIRQV (X)
55308	$080C	PDNAME (X)
55298	$0802	PDREVN (X)
55300	$0804	PDTYPE (X)
53504	$D100	PDVIN
53759	$D1FF	PDVIRQ (X)
583	$247	PDVMSK (X)
53504	$D100	PDVOUT
53505	$D101	PDVRST
53505	$D101	PDVSTA
53759	$D1FF	PDVS
55309	$D80D	PDVV (X)
54284	$D40C	PENH
54285	$D40D	PENV
58502	$E486	PHENTV (X)
58508	$E48C	PHINIV (X)
58505	$E489	PHULNV (X)
56640	$DD40	PLYEVL

54279	$D407	PMBASE
16	$10	POKMSK
54016	$D300	PORTA
54017	$D301	PORTB (A)
54017	$D301	PORTB (X)
53760	$D200	POT0
53761	$D201	POT1
53762	$D202	POT2
53763	$D203	POT3
53764	$D204	POT4
53765	$D205	POT5
53766	$D206	POT6
53767	$D207	POT7
53771	$D208	POTGO
588	$24C	PPTMPA (X)
589	$240	PPTMPX (X)
58416	$E430	PRINTV
53275	$D01B	PRIOR
960-999	$3C0-$3E7	PRNBUF
201	$C9	PTABW
31	$1F	PTEMP (A)
28	$1C	PTIMOT (A)
788	$314	PTIMOT (X)
636	$27C	PTRIG0
637	$27D	PTRIG1
638	$27E	PTRIG2
639	$27F	PTRIG3
640	$280	PTRIG4
641	$281	PTRIG5
642	$282	PTRIG6
643	$283	PTRIG7
829	$33D	PUPBT1 (X)
830	$33E	PUPBT2 (X)
831	$33F	PUPBT3 (X)
58496	$E480	PUPDIV (X)
65530,65531	$FFFA,$FFFB	PVECT

251	$FE	RADFLG
4,5	$4,55	RAMLO
740	$2E4	RAMSIZ
106	$6A	RAMTOP
53770	$D20A	RANDOM
58490	$E47A	RBLOKV
581	$245	RECLEN (X)
57	$39	RECVDN
586,587	$24A,$248	RELADR (X)
83	$53	RMARGN
112,113	$70,$71	ROWAC
84	$54	ROWCRS
121	$79	ROWINC (A)
760	$2F8	ROWINC (X)
18,19,20	$12,$13,$14	RTCLOK
713,714	$2C9,$2CA	RUNADR (X)
736,737	$2E0,$2E1	RUNAD
142,143	$8E,$8F	RUNSTK
104,105	$68,$69	SAVADR
790	$316	SAVIO
8E1,89	$58,$59	SAVMSC
58384	$E410	SCRENV
699	$288	SCRFLG
560,561	$230,$231	SDLSTL
559	$22F	SDMCTL
58472	$E468	SENDEV
53773	$D20D	SERIN
53773	$D20D	SEROUT
58460	$E450	SETVBV
111	$6F	SHFAMT
702	$28E	SHFLOK
584	$248	SHPDVS (X)
58469	$E465	SIOINV
58457	$E459	SIOV
53260	$D00C	SIZEM
53256	$D008	SIZERO

53257	$D009	SIZEP1
53258	$D00A	SIZEP2
53259	$D00B	SIZEP3
53775	$020F	SKCTL
53770	$D20A	SKREST
53775	$D20F	SKSTAT
58499	$E483	SLFTSV (X)
65	$41	SOUNDR
555	$228	SRTIMR
767	$2FF	SSFLAG
562	$232	SSKCTL
792	$318	STACKP
140,141	$8C,$80	STARE
48	$30	STATUS
632	$278	STICK0
633	$279	STICK1
634	$27A	STICK2
635	$278	STICK3
53769	$D209	STIMER
138,139	$8A,$8B	STMCUR
136,137	$88,$89	STMTAB
186,187	$BA,$BB	STOPLN
20480	$5000	STORG
670	$29E	SUBTMP
1000	$3E8	SUPERF (X)
123	$78	SWPFLG
58463	$E45F	SYSVBV
675-689	$2A3,$281	TABMAP
574	$23E	TEMP
780,781	$30C,$30D	TIMER1
784,785	$310,$311	TIMER2
791	$317	TIMFLG
659	$293	TINDEX
80	$50	TMPCHR
697,698	$289,$28A	TMPCOL
673	$2A1	TMPLBT

696	$2BB	TMPROW
668	$29C	TMPX1 (A)
6	$6	TRAMSZ (A)
53264	$D010	TRIG0
53265	$D011	TRIG1
53266	$D012	TRIG2
53266	$D012	TRIG2 (XEGS)
53267	$D013	TRIG3 (A)
53267	$D013	TRIG3 (X)
6	$6	TRNSMZ (X)
793	$319	TSTAT
7	$7	TSTDAT (A)
7	$7	TSTDAT (X)
657,658	$291,$292	TXTCOL
660,661	$294,$295	TXTMSC
662-667	$296-$29B	TXTOLD
656	$290	TXTROW
518,519	$206,$207	VBREAK
54283	$D40B	VCOUNT
53276	$D01C	VDELAY
512,513	$200,$201	VDSLST
534,535	$216,$217	VIMIRQ
516,517	$204,$205	VINTER
520,521	$208,$209	VKEYBD
132,133	$84,$85	VNTD
130,131	$82,$83	VNTP
568,569	$238,$239	VPIRQ (X)
514,515	$202,$203	VPRCED
54277	$D405	VSCROL
522,523	$20A,$208	VSERIN
526,527	$20E,$20F	VSEROC
524,525	$20C,$20D	VSEROR
620	$26C	VSFLAG (X)
528,529	$210,$211	VTIMR1
530,531	$212,$213	VTIMR2
532,533	$214,$215	VTIMR4

548,549	$224,$225	VVBLKD
546,547	$222,$223	VVBLKI
134,135	$86,$87	VVTP
8	$8	WARMST
58484	$E474	WARMSV
649	$289	WMODE
54282	$D40A	WSYNC
58466	$E462	XITVBV
58	$3A	XMTDON
67,68	$43,$44	ZDUFP
74,75	$4A,$4B	ZCHAIN (X)
69,70	$45,$46	ZDRVA
55876	$DA44	ZFR0
55878	$DA46	ZFR1
717,718	$2CD,$2CE	ZHIUSE (X)
723,724	$2D3,$2D4	ZLOADA (X)
71,72	$47,$48	ZSBA
245,246	$F5,$F6	ZTEMP1
247,248	$F7,$F8	ZTEMP2
249,250	$F9,$FA	ZTEMP3

4 Anschlüsse

Die hier beschriebenen technischen Informationen beziehen sich zumeist auf die NTSC- und PAL-Versionen der ATARI-8-Bit-Computer.

Als Referenz stehen dank Curt Vendel (www.atarimuseum.com) einige Originaldokumente von ATARI zur Verfügung. Die darin enthaltenen Informationen weichen zum Teil von den in der Vergangenheit in Magazinen, Büchern und auf Internetseiten publizierten Inhalten ab. In diesem Kapitel wird der Versuch gewagt, möglichst eindeutig darzulegen, welche Pin-Belegungen, Anschlüsse, etc. wo zu finden sind.

Intern
ATARI hatte neben handelsüblichen Bauteilen auch speziell für die ATARI-Computer entwickelte Chips sowie programmierbare Logikbausteine verwendet. Manche Handelsübliche und alle ATARI-spezifischen Chips sind mit hauseigenen "CO"-Nummern gekennzeichnet. Da die Kennzeichnung aber nicht durchgängig ist und im Laufe der Jahrzehnte bei Reparaturen und Umbauten eventuell ein Austausch von Chips erfolgt sein könnte, wendet euch bei Unklarheiten mit eurer Hardware an den ABBUC e.V.

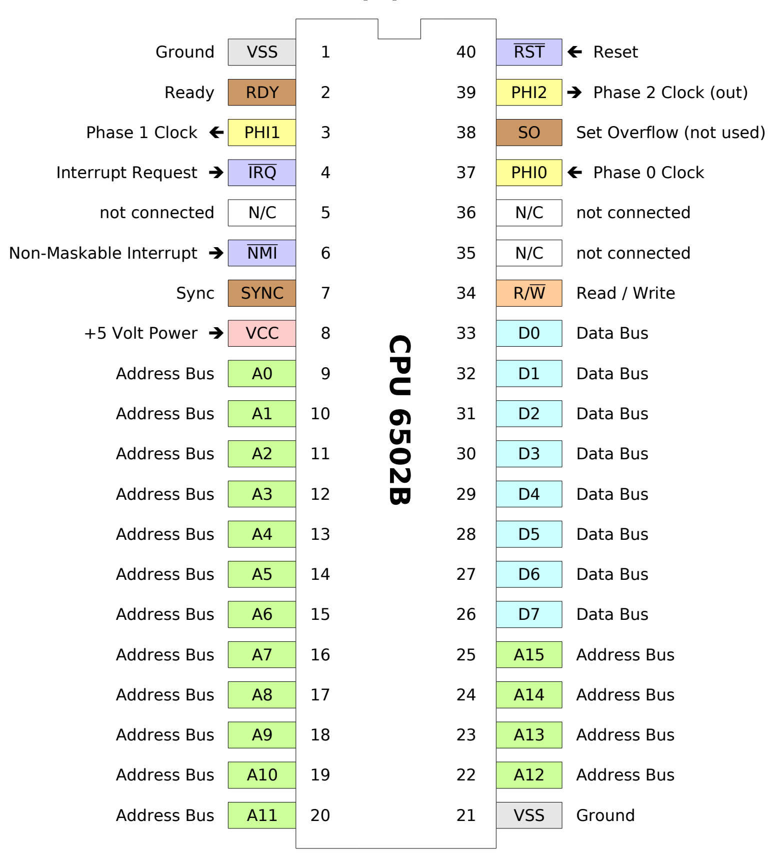
Die relevanten Chips im ATARI sind:

CPU
ANTIC
GTIA
POKEY
PIA
FREDDIE (XL/XE)
MMU (XL/XE)

Extern
Die relevanten Anschlüsse nach außen sind:

SIO-Port
Controller-Port (auch Joystickport genannt)
Cartridge Port (Slot)
Parallel Bus Interface → 600XL, 800XL
Enhanced Cartridge Interface → 65XE, 800XE, 130XE
Monitorbuchse
TV-Buchse
Buchse Stromversorgung
Tastaturbuchse → XEGS

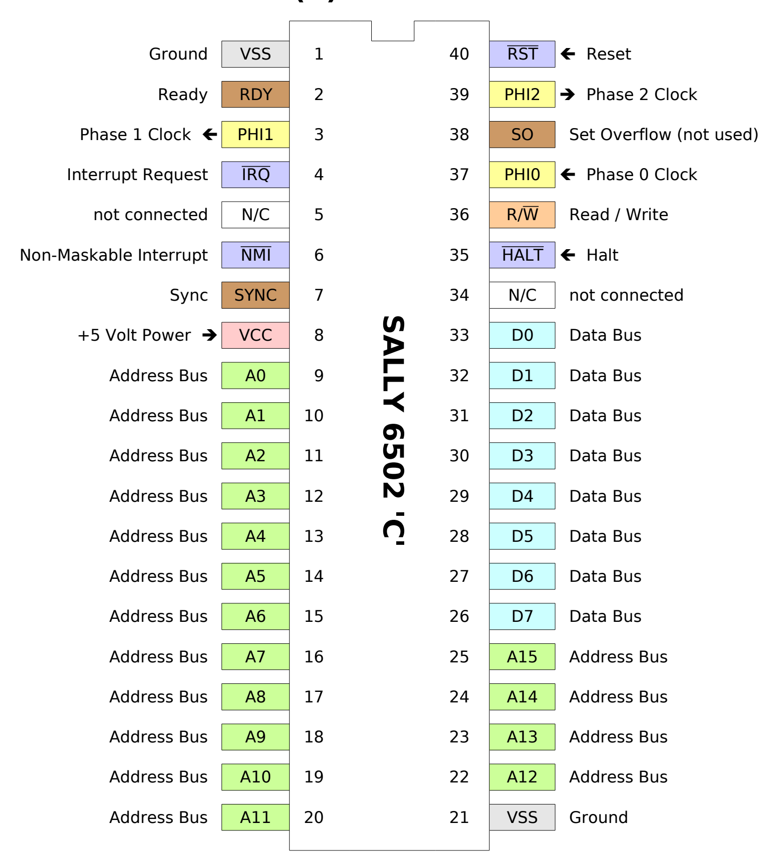
4.1 CPU im 400/800 (A)



CO14377 (NTSC), ggf. auch nur mit Herstellerbezeichnung bedruckt, z.B. MCS6502. Die bisher untersuchten PAL-400/800-ATARIs enthielten statt der Standard-6502B-CPU bereits eine SALLY.

PHI 0	Takteingang der CPU
PHI 1	invertierter Taktausgang
PHI 2	nicht invertierter Taktausgang
IRQ	Interrupt Request – siehe Sally CPU.
NMI	nicht maskierbarer Interrupt – siehe Sally CPU.
RESET	L initialisiert CPU – siehe Sally CPU

4.2 CPU 'SALLY' (X)

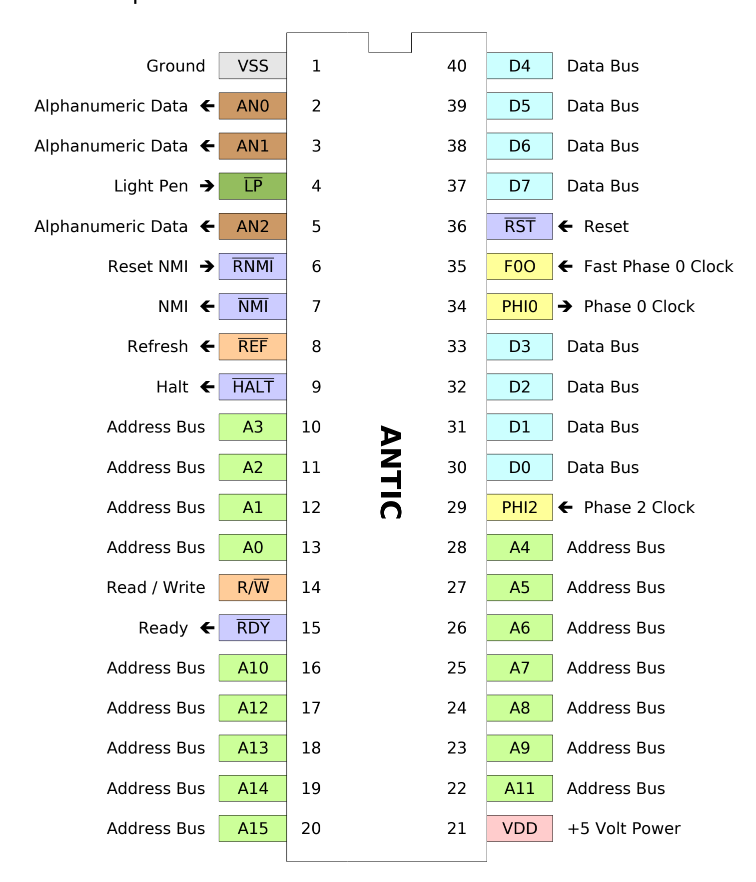


CO14806 (PAL/NTSC/SECAM) → auch im 400/800 (PAL) zu finden !
Höchstwahrscheinlich ist in der SALLY eine 6502B-CPU enthalten.

PHI 0	Takteingang der CPU
PHI 1	invertierter Taktausgang
PHI 2	nicht invertierter Taktausgang
$\overline{\text{IRQ}}$	Interrupt Request. L löst einen Interrupt aus (falls das Interrupt-Flag der CPU nicht gesetzt ist), d.h. die CPU setzt ihren Programmzähler auf den Inhalt von $FFFE,$FFFF
$\overline{\text{NMI}}$	nicht maskierbarer Interrupt. L löst einen Interrupt aus, d.h., die CPU setzt ihren Programmzähler auf den Inhalt von $FFFA,$FFFB
$\overline{\text{HALT}}$	Hält CPU an.
$\overline{\text{RESET}}$	L initialisiert CPU und setzt Programmzähler auf $FFFC,$FFFD.

4.3 ANTIC

ANTIC = **A**lpha**n**umeric **T**elevision **I**nterface **C**ontroller

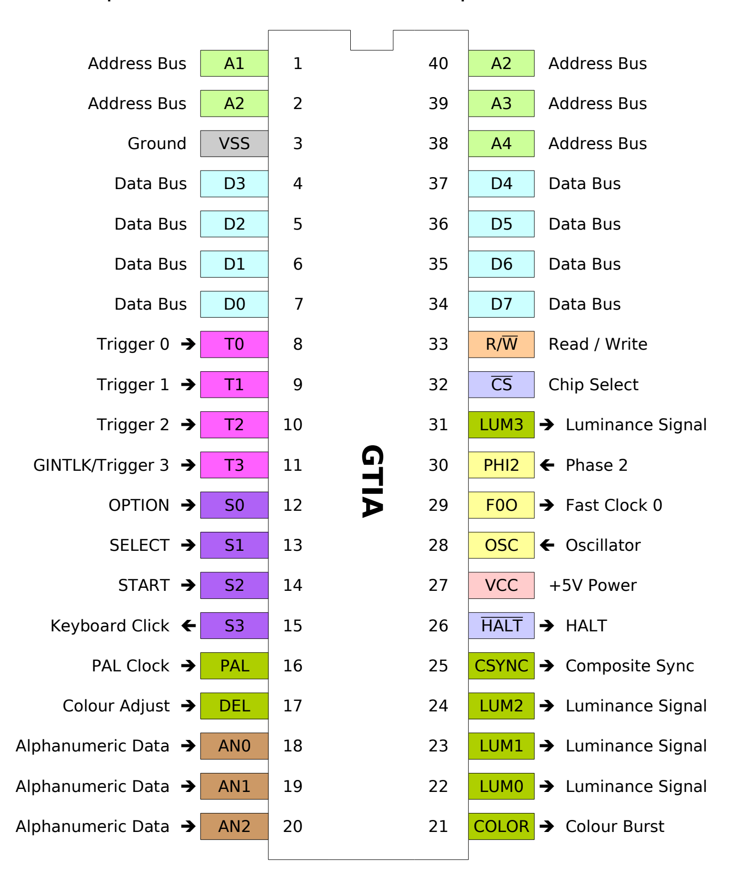


400/800/XL/XE	CO14887, CO21698 (PAL) CO12296; CO21697 (NTSC)

AN0-AN2	GTIA Data Output
RDY	Memory Ready
Reset NMI	Erzeugt den System-Reset-NMI beim ATARI 400/800

4.4 GTIA

GTIA = **G**raphic **T**elevision **I**nterface **A**daptor

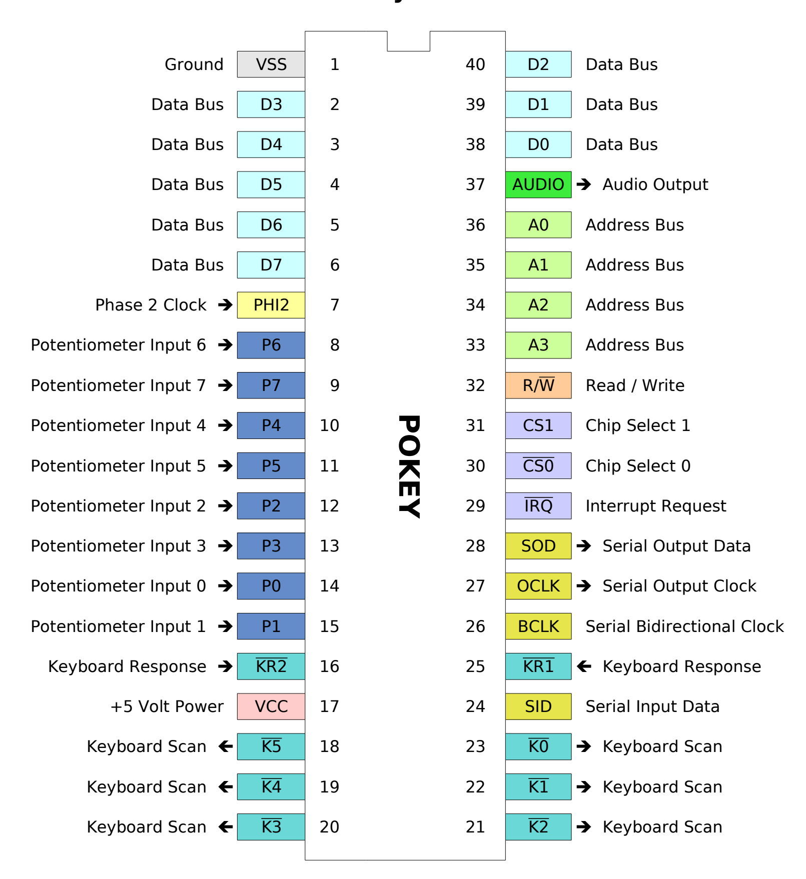


400/800/XL/XE CO14889 (PAL), CO14805 (NTSC), CO20120 (SECAM)
CO12295 (NTSC) ist Vorgänger CTIA (*Color* TIA).

DEL Color Delay Line Adjustment (manchmal als CADJ bezeichnet)
HALT synchronisiert ANTIC & GTIA (wenn Low und im Horizontal Blank)
PAL dieser Pin ist nur in der PAL-Version belegt.
S0-S2 sind vom Betriebssystem als Eingänge programmiert und mit den Konsole-Tasten (→ CONSOL, $D01F) verbunden.
S3 erzeugt Tastaturklick

4.5 POKEY

POKEY = **Po**tentiometer and **Key**board Controller

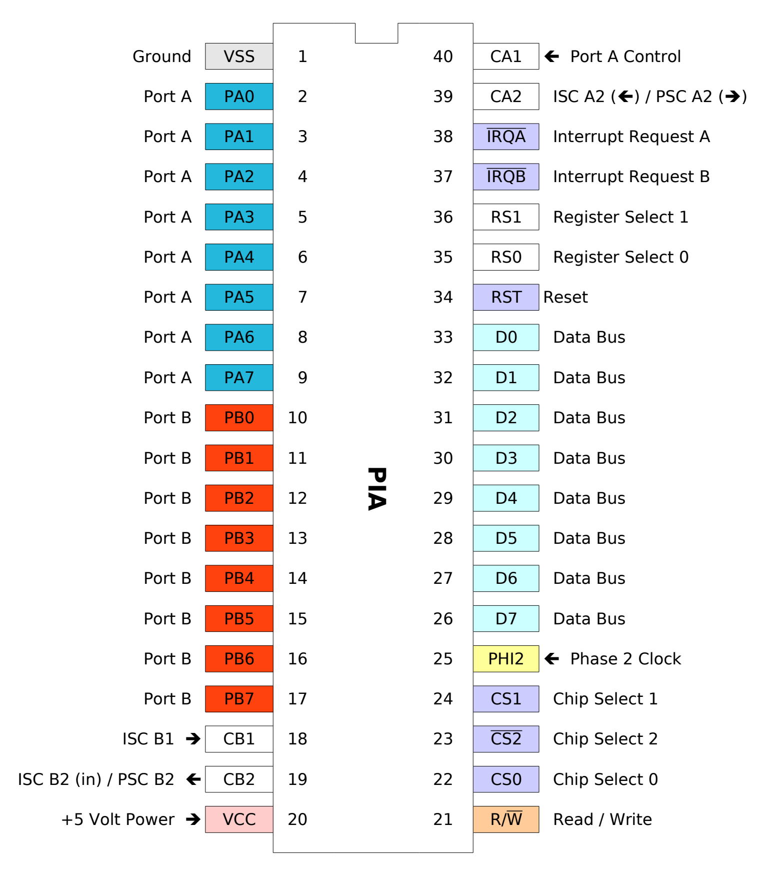


400/800/XL/XE CO12294 (PAL/NTSC)

K0-K5 6-Bit-Wert (0-63; $00-$3F) zum Dekodieren von 64 Tasten.
KR1 Der 6-Bit-Wert der gedrückten Taste wird hier zurückgeliefert.
KR2 Dient zum Dekodieren der Tasten SHIFT, BREAK und CONTROL.
P0-7 Lesen Potentiometerwerte von 0-228 ($0-$E4).

4.6 PIA

PIA = **P**eripheral **I**nterface **A**dapter

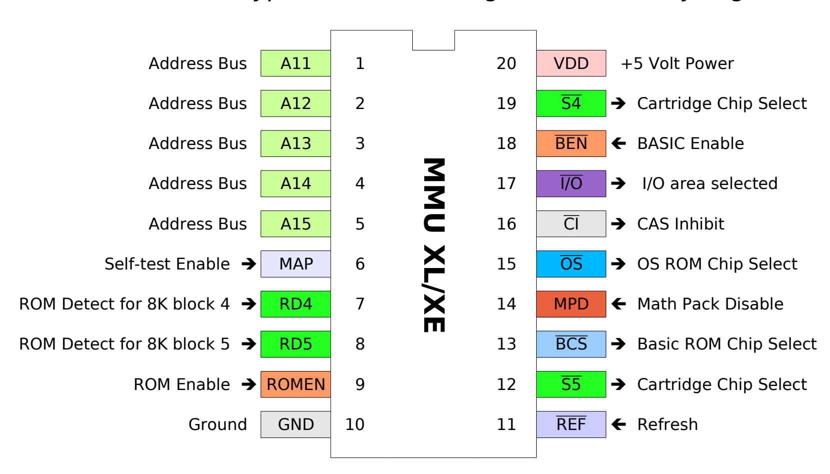


400/800/XL/XE CO14795, CO14812 (PAL/NTSC)
Oft ohne CO-Nummer, da handelsübliche 6520 (6521, 6820, 6821).

PA0-PA3	bidirektionaler Controller Port 1
PA4-PA7	bidirektionaler Controller Port 2
PB0-PB7	bidirektionaler Port B (beim XL sind Bit 2-6 unbenutzt)
CA1	SIO Proceed
CA2	SIO Motor Control
CB1	SIO Interrupt
CB2	SIO Command

4.7 MMU XL/XE

PAL-Baustein vom Typ 16L8 (PAL=Programmable Array Logic)

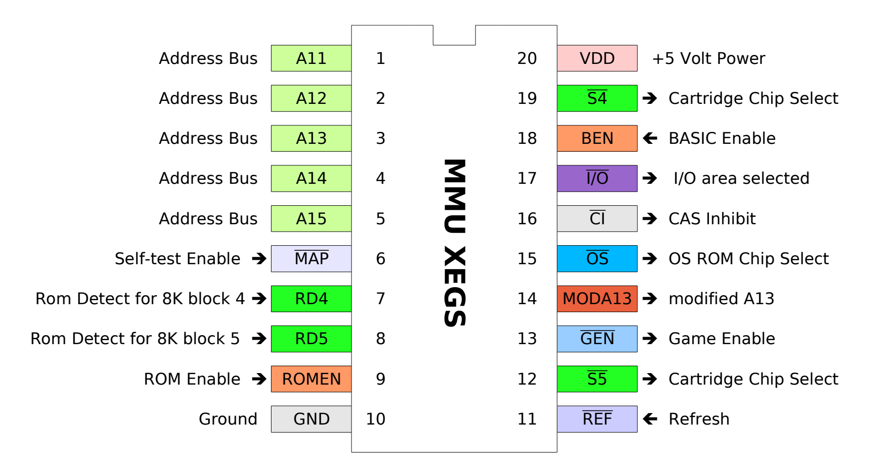


CO61618

MAP	von PIA PB7: Selbsttest, $5000-$57FF, L=ein, H=aus
RD4	vom Modulport: H=Modul will Speicher (meist ROM) bei $8000-$9FFF einblenden
RD5	vom Modulport: H=Modul will Speicher (meist ROM) bei $A000-$BFFF einblenden
ROMEN	von PIA PB0 – Betriebssystem, L=OS aus, H=OS ein
REF	vom ANTIC erzeugtes Refresh-Signal für die dynamischen RAMs
BCS	BASIC-ROM Chip Select
MPD	vom PBI wird Mathe-ROM abgeschaltet und das ROM des PBI-Geräts an $D800-$DFFF eingeblendet.
OS	OS-ROM Chip Select
CI	deaktiviert das RAM
IO	geht an 74LS138, selektiert Hardware-Chips → $D0xx - $D7xx (Y0 - GTIA, Y2 - POKEY, Y3 - PIA, Y5 – CCTL), decodiert den Adressbereich von $D000 - $D7FF
BEN	von PIA PB1 – internes BASIC, L=ein, H=aus
S4	zum Modulport: L=CPU greift auf $8000-$9FFF zu und RD4 ist H, d.h., Modul soll seinen Speicher (ROM) aktivieren
S5	zum Modulport: L=CPU greift auf $A000-$BFFF zu und RD5 ist H, d.h., Modul soll seinen Speicher (ROM) aktivieren

4.8 MMU XEGS

PAL-Baustein vom Typ 16L8



C101686

Hier werden nur die Unterschiede zur XL/XE-MMU aufgezeigt.

MODA13 zum ROM: modifiziertes A13, um das im 32KB-ROM enthaltene BASIC bzw. Missile Command in den richtigen Speicherbereich \$A000-\$BFFF einzublenden

$\overline{\text{GEN}}$ von PIA, PB6: L aktiviert Missile Command im 32KB-ROM des XEGS

Das XEGS-Betriebssystem kann über TRIG2 (53266; \$D012) des GTIA prüfen, ob die Tastatur angeschlossen ist. Wenn nicht, setzt es beim Einschalten PB6 der PIA auf Low und aktiviert damit das im ROM eingebaute Missile Command.

Im 32 KB großen ROM des XEGS sind Betriebssystem (XE), BASIC und das Spiel "Missile Command" integriert.

4.9 Freddie (X)



CO61921, CO61922, CO61991 (PAL/NTSC); verwendet in XLF, XE, XEGS.

Der Freddie ersetzt einige ICs zur Takterzeugung und Ansteuerung der dynamischen RAMs im ATARI.

CLKIN	Takteingang (14,18757/14,31818 MHz PAL/NTSC) (bei bestücktem Quarzoszillator mit dessen Ausgang verbunden, bei bestücktem Quarz mit dem einen Pin des Quarzes verbunden)
CLKOUT	Taktausgang (MHz s.o.) (bei bestücktem Quarzoszillator unbenutzt, bei bestücktem Quarz mit dem anderen Pin des Quarzes verbunden)
EXTSEL	External Select Input vom Parallelbus.

LE, OE und 16KCAS sind unbenutzt.

4.10 ATARI 400

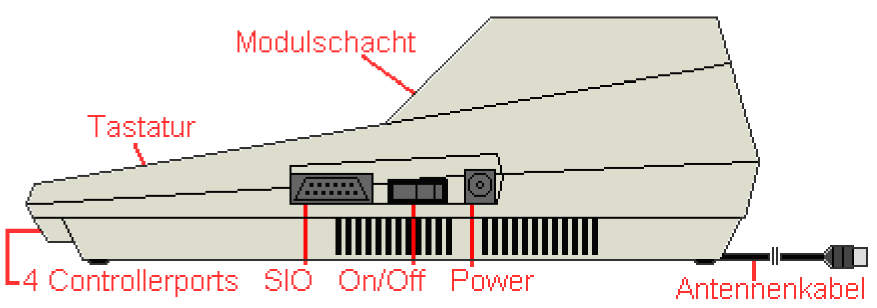


Abb. 4.1: Anschlüsse 400

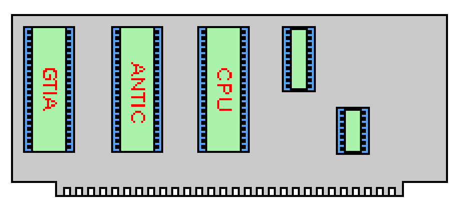


Abb. 4.2: CPU-Board 400 (PAL)

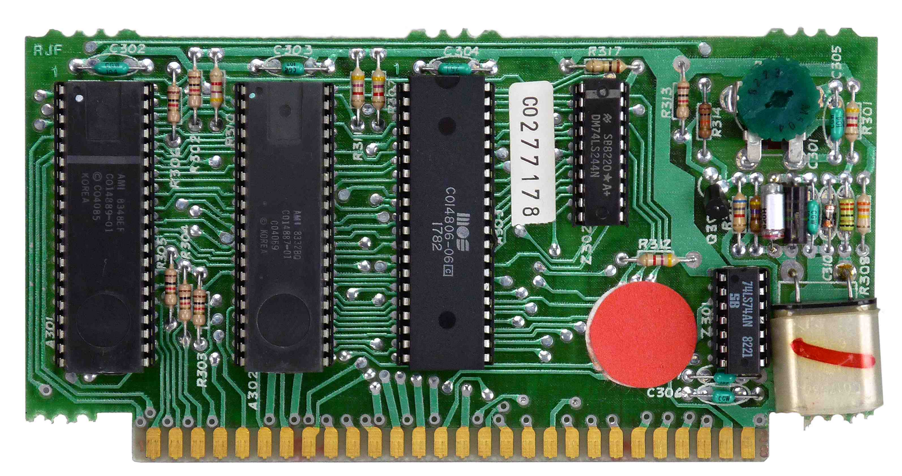


Abb. 4.3: Foto CPU Board 400 (PAL)



Abb. 4.4: Foto RAM Board 400 (16KByte)

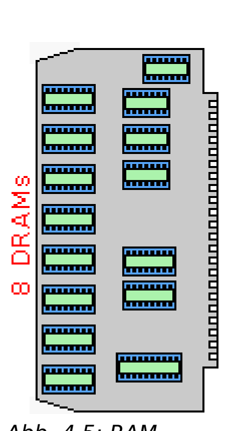


Abb. 4.5: RAM Board 400 (48KByte)

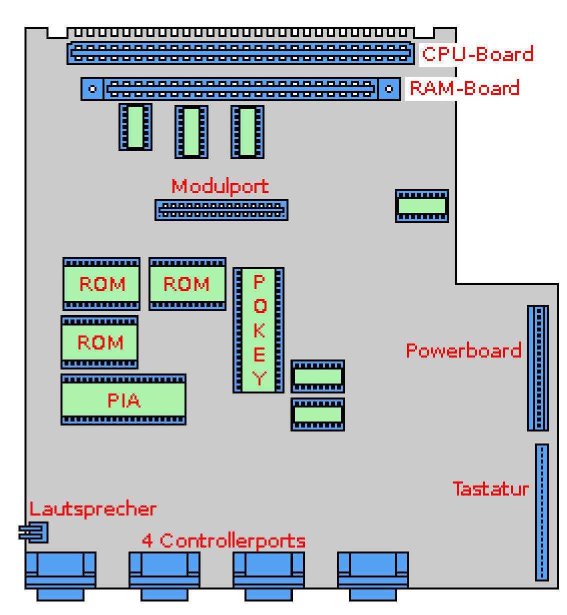


Abb. 4.6: Mainboard 400



Abb. 4.7: Foto Mainboard 400

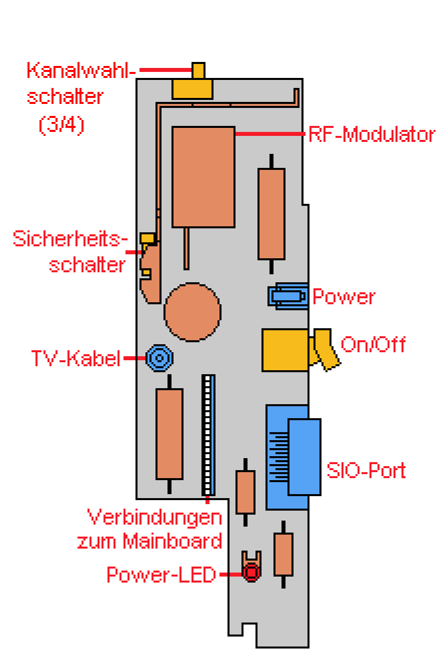


Abb. 4.8: Power-Board 400

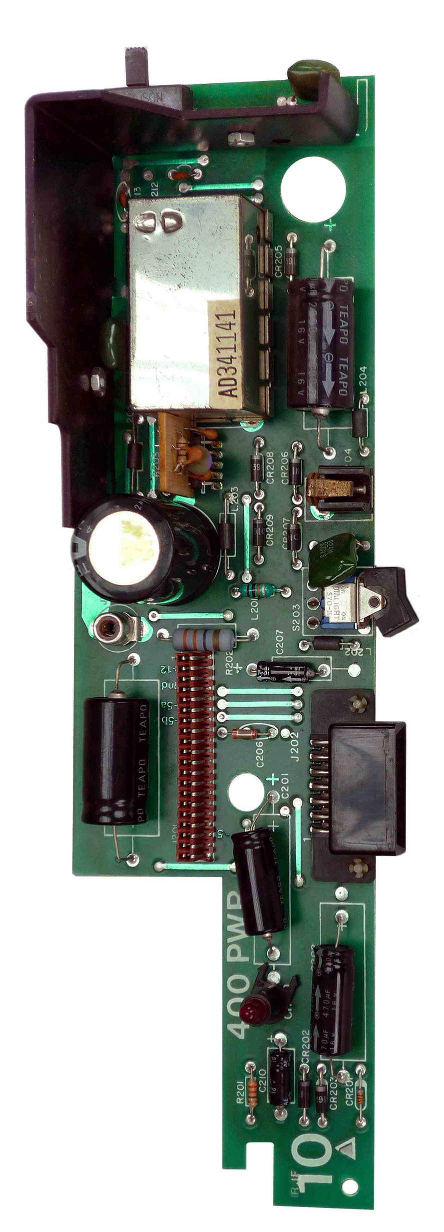


Abb. 4.9: Foto Power Board 400

4.11 ATARI 800

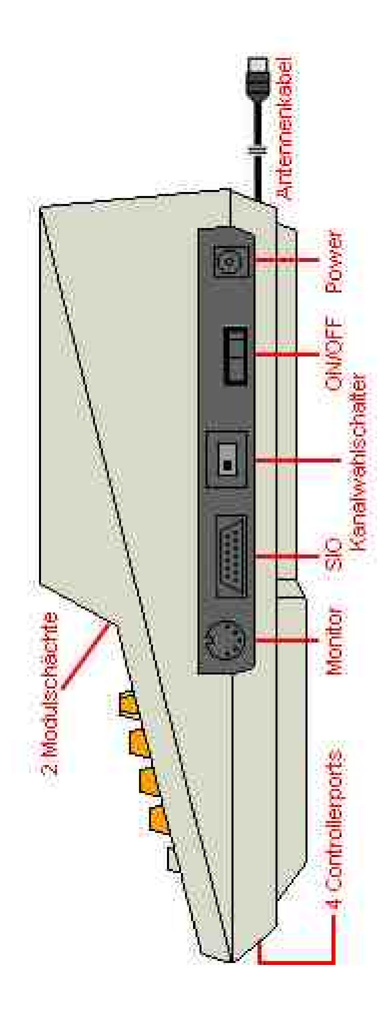


Abb. 4.10: Anschlüsse 800

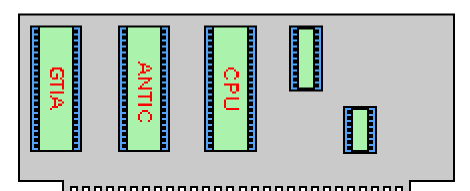


Abb. 4.11: CPU-Board 800 (PAL)



Abb. 4.12: Foto CPU-Board 800 (PAL) mit CPU 'SALLY'

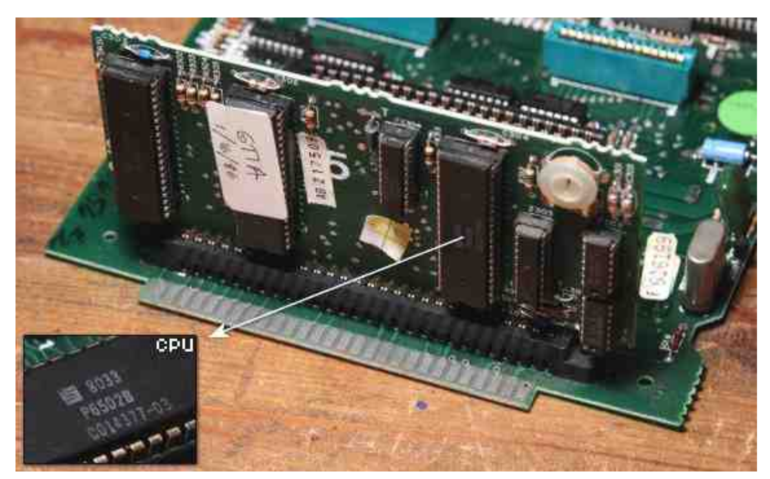


Abb. 4.13: Foto CPU-Board 800 (NTSC) mit CPU 6502B

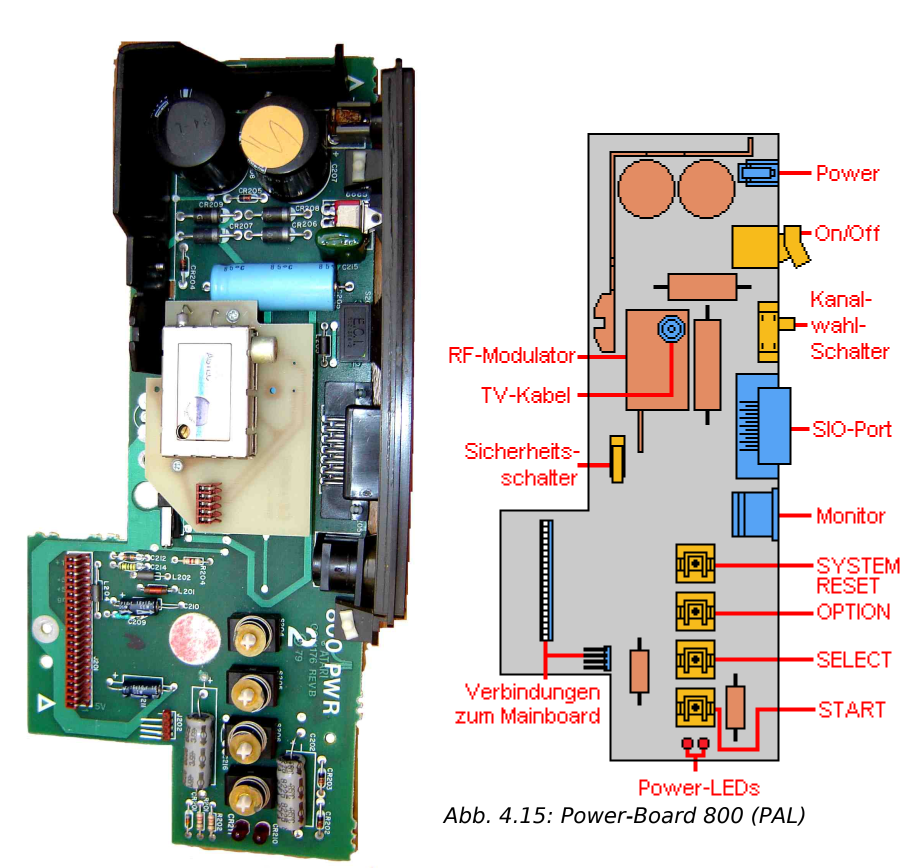


Abb. 4.14: Foto Power-Board 800 (PAL)

Abb. 4.15: Power-Board 800 (PAL)

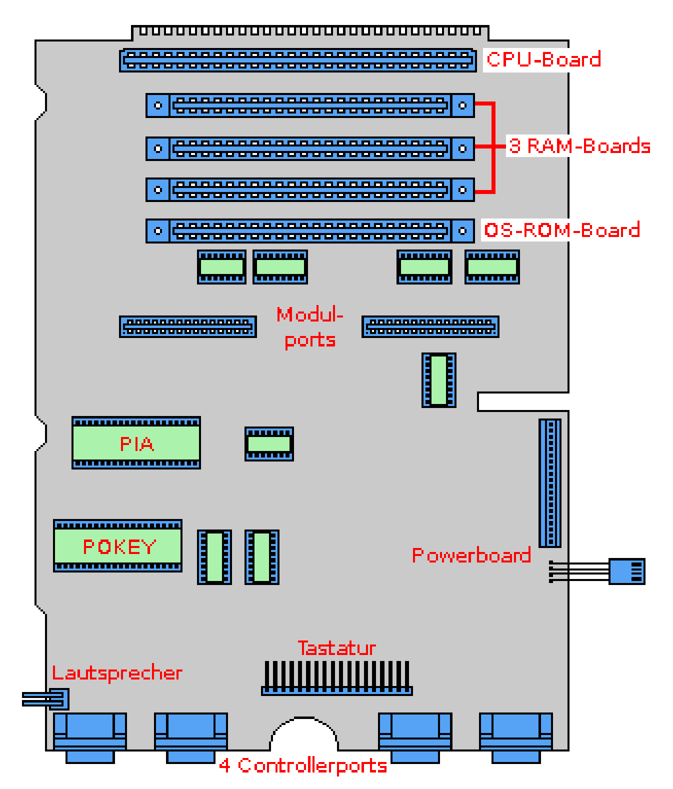


Abb. 4.16: Mainboard 800

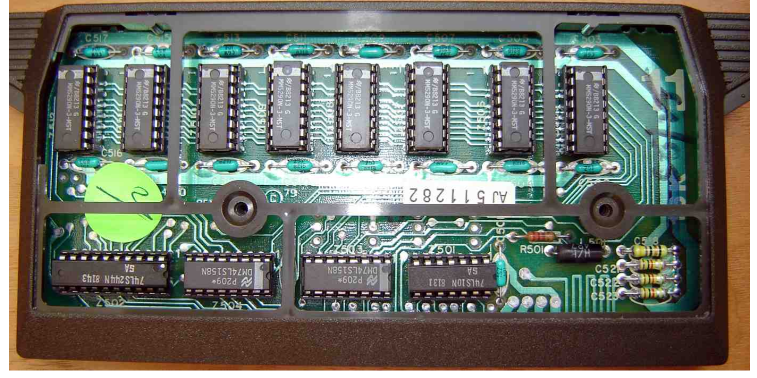


Abb. 4.17: Foto 16 KByte Memory Board für 800

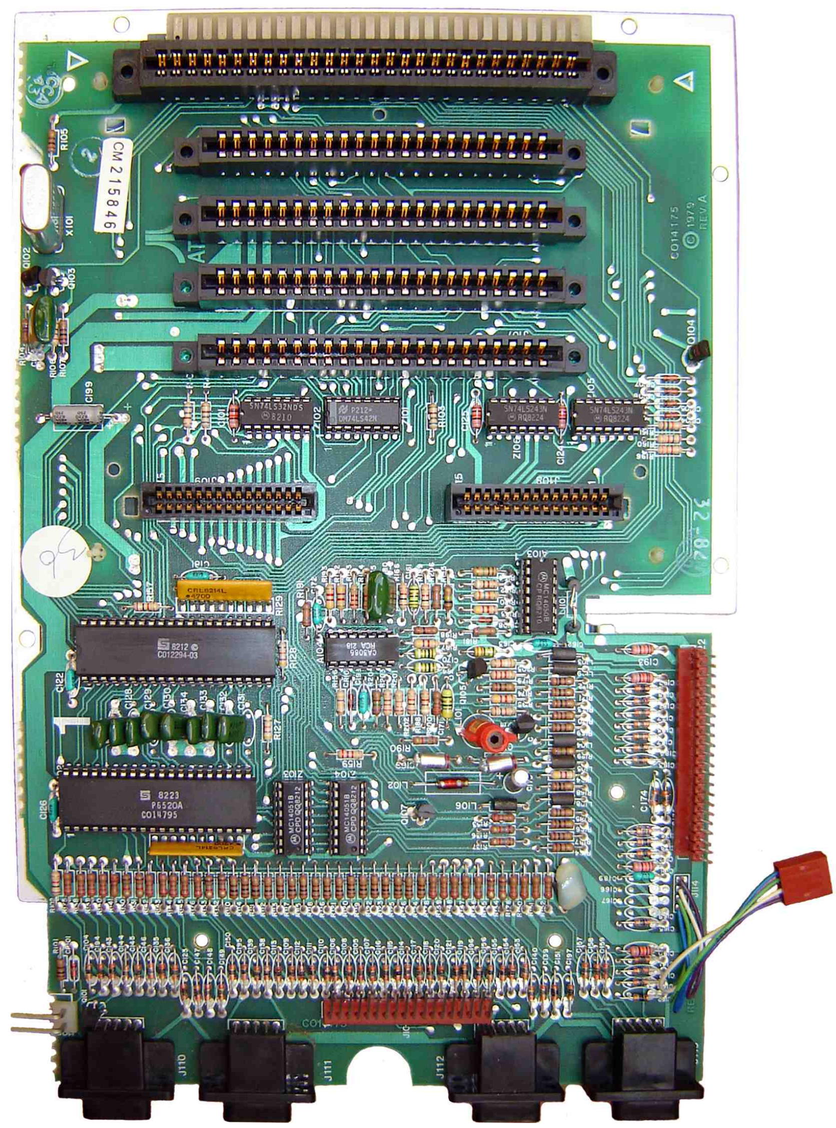


Abb. 4.18: Foto Mainboard 800 (PAL)

4.12 ATARI 1200XL

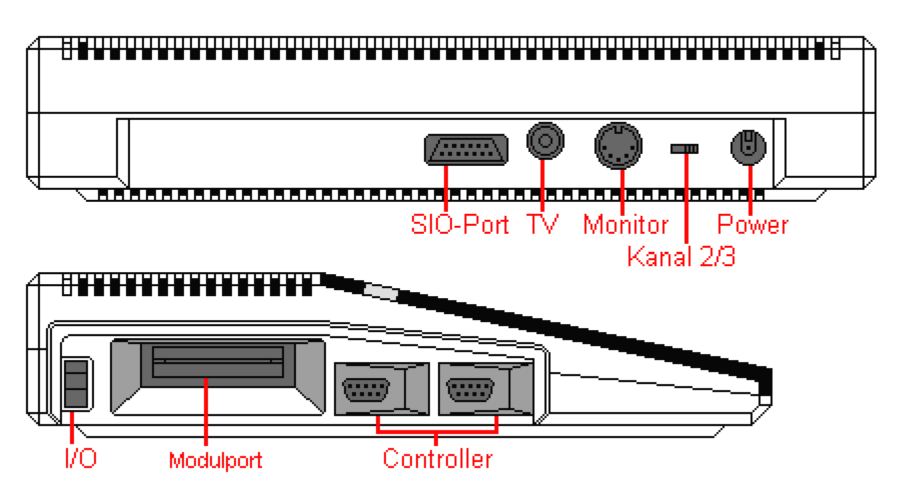


Abb. 4.19: Anschlüsse 1200XL (NTSC)



Abb. 4.20: Tastatur 1200XL mit LED und Funktionstasten

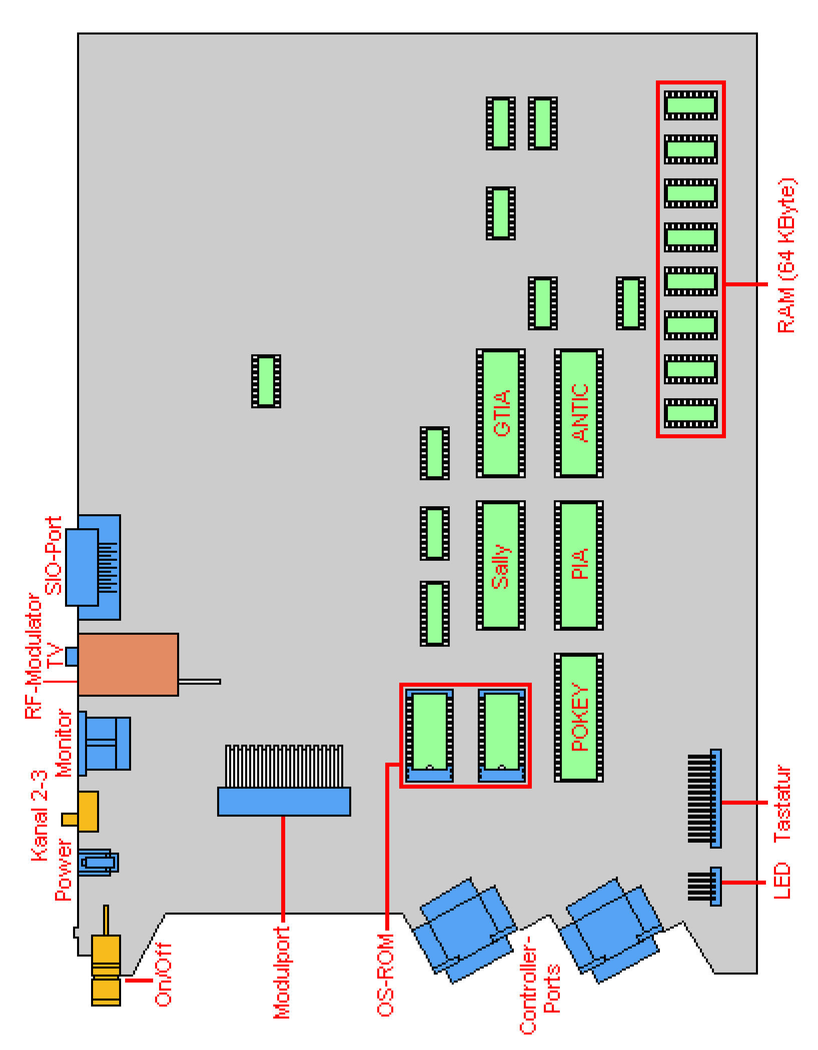


Abb. 4.21: Mainboard 1200XL (NTSC)



Abb. 4.22: Foto Mainboard 1200XL (NTSC)

4.13 ATARI 600XL

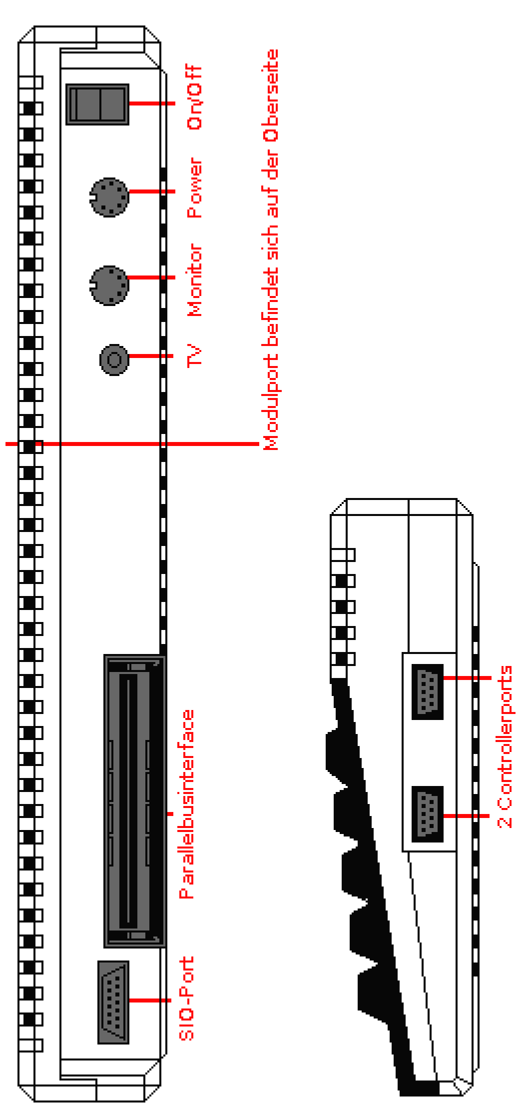


Abb. 4.23: Anschlüsse 600XL (PAL)

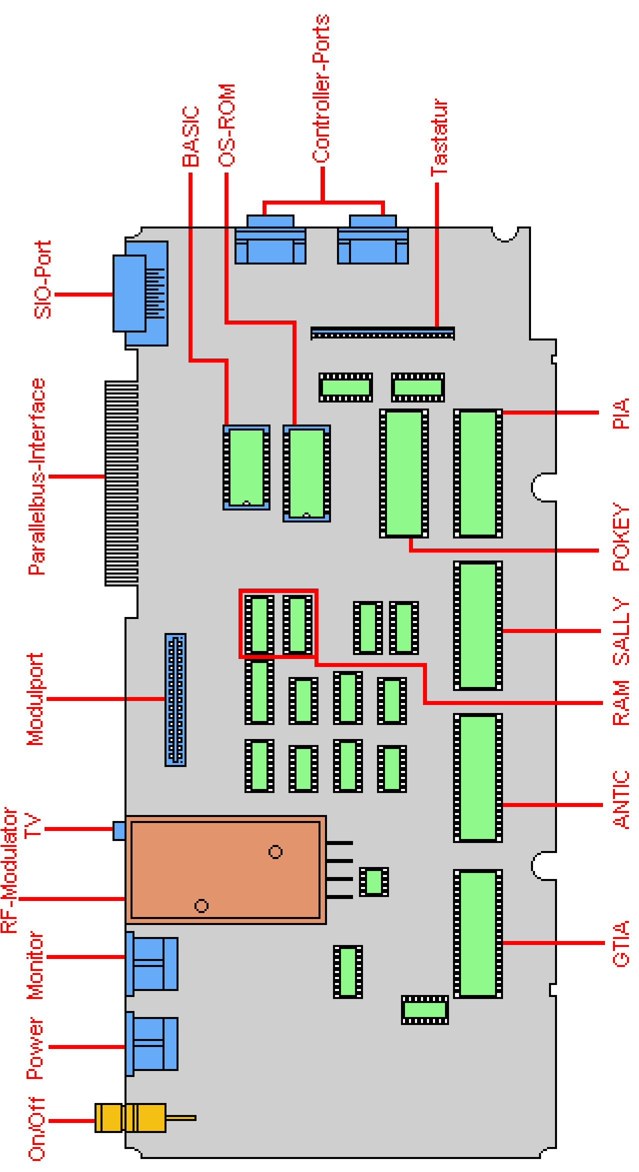


Abb. 4.24: Mainboard 600XL (PAL)

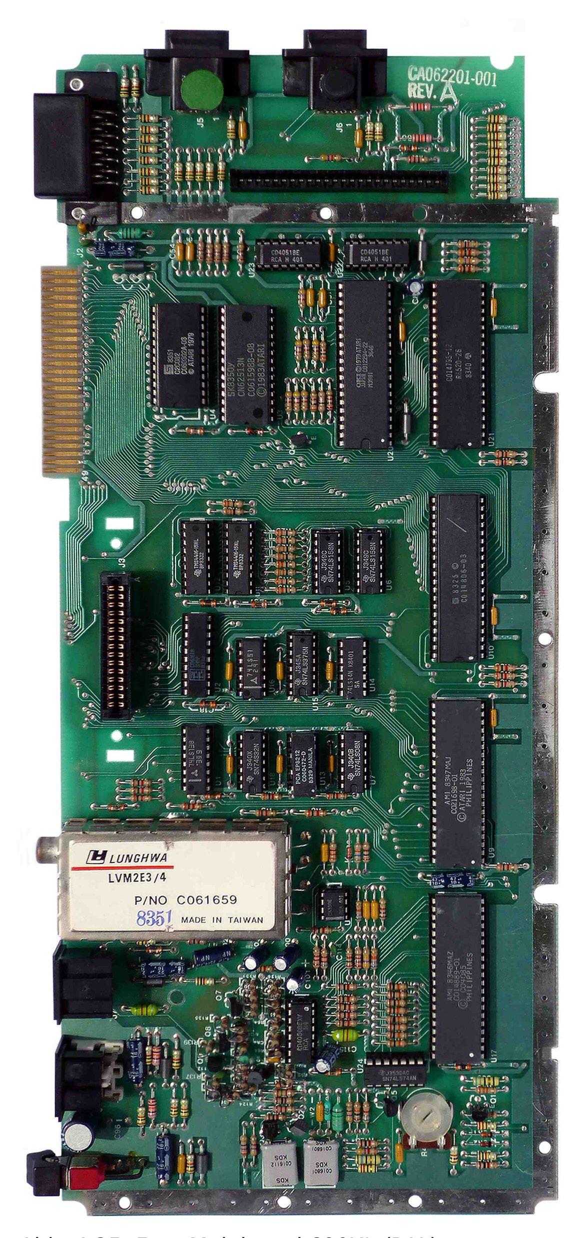


Abb. 4.25: Foto Mainboard 600XL (PAL)

4.14 ATARI 800XL/XLF

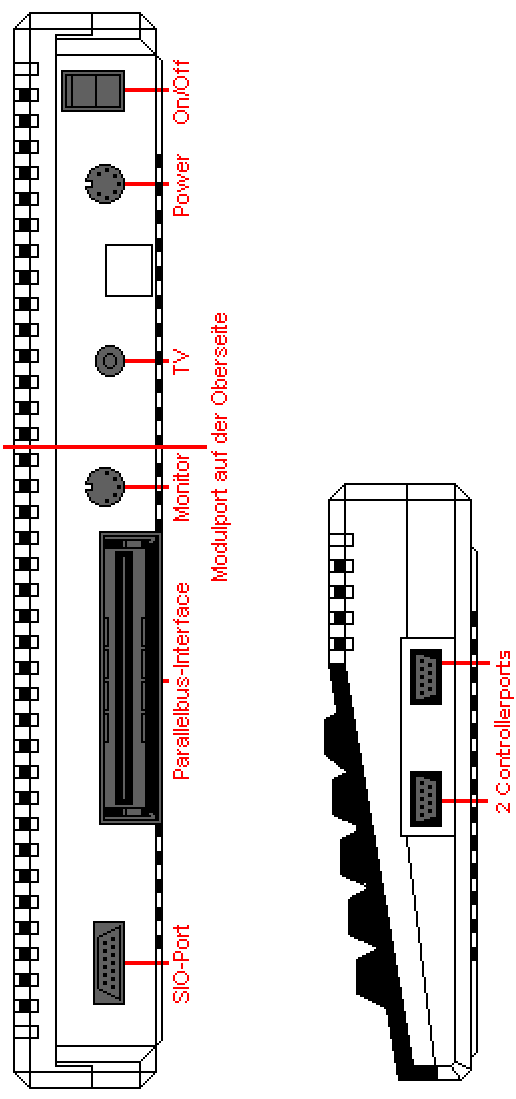


Abb. 4.26: Anschlüsse 800XL/XLF (PAL)

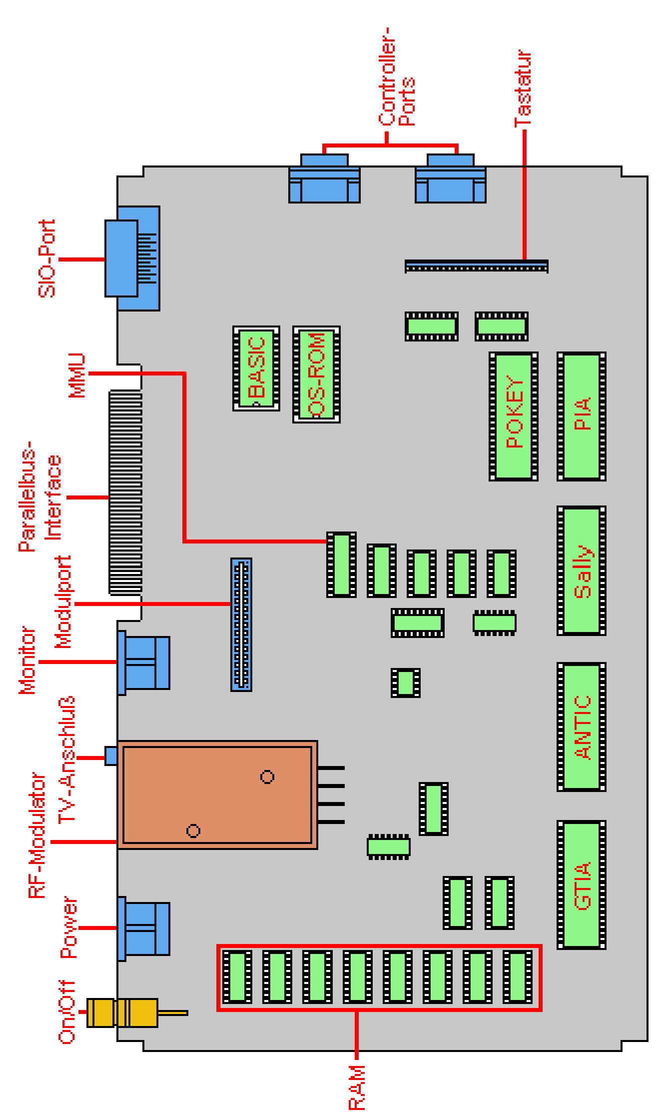


Abb. 4.27: Mainboard 800XL (PAL)



Abb. 4.28: Foto Mainboard 800XL (PAL)

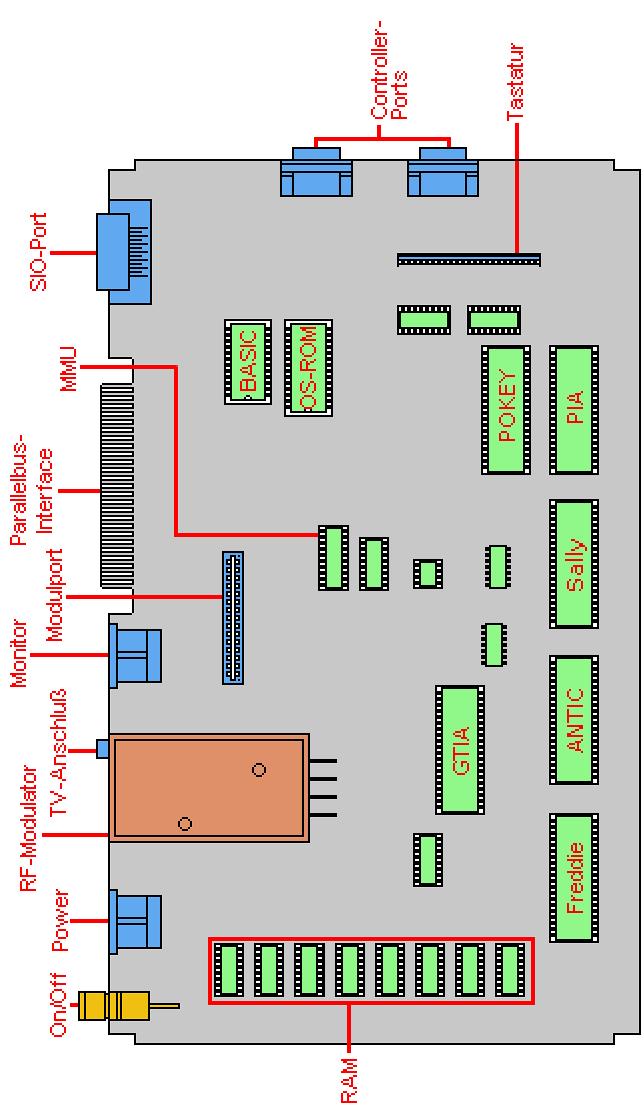


Abb. 4.29: Mainboard 800XLF "Rose" (PAL)

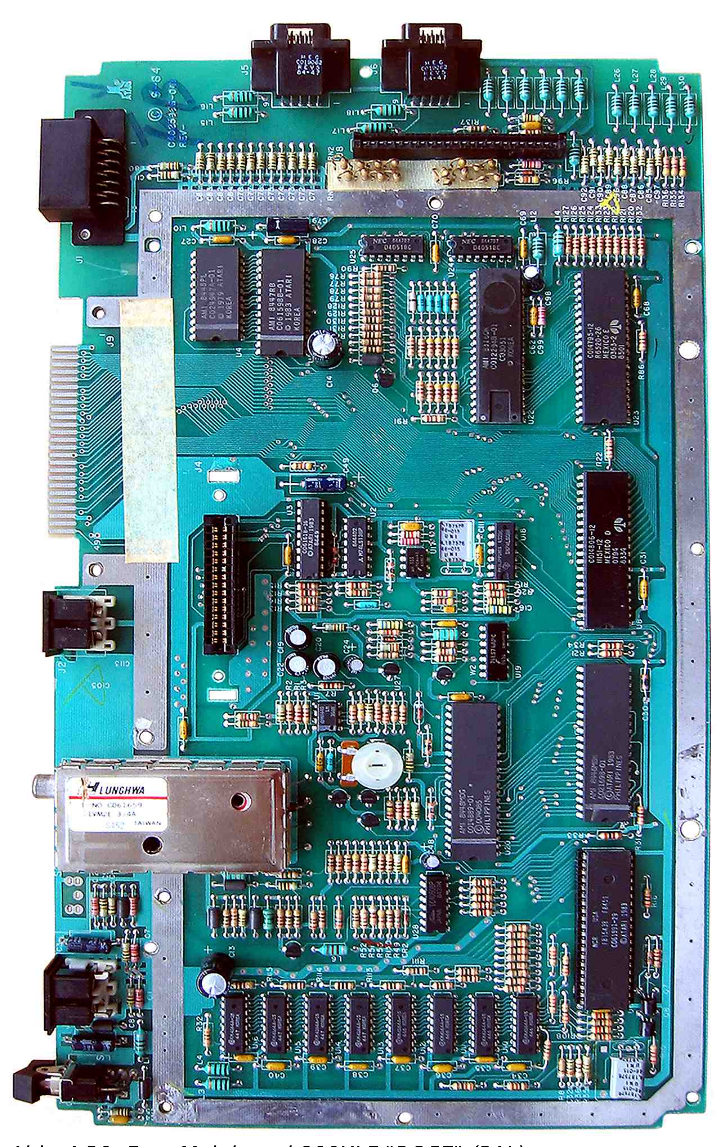


Abb. 4.30: Foto Mainboard 800XLF "ROSE" (PAL)

4.15 ATARI 65XE/800XE/130XE

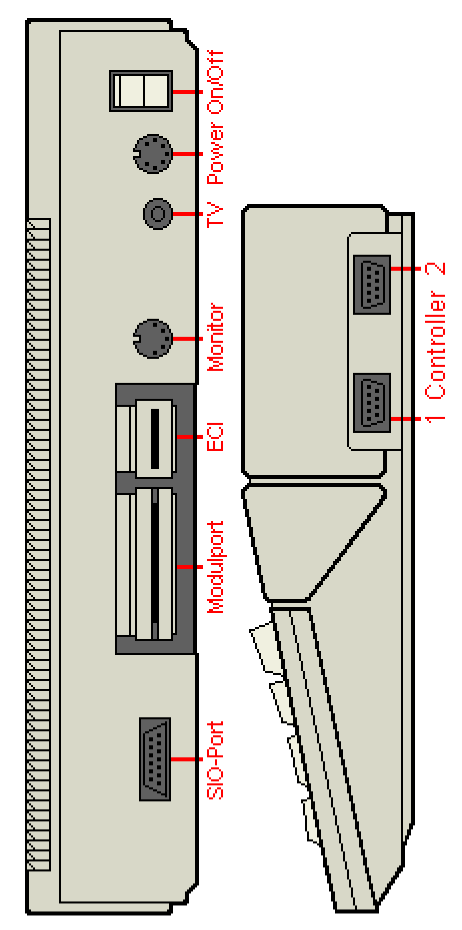


Abb. 4.31: Anschlüsse XE-Modelle

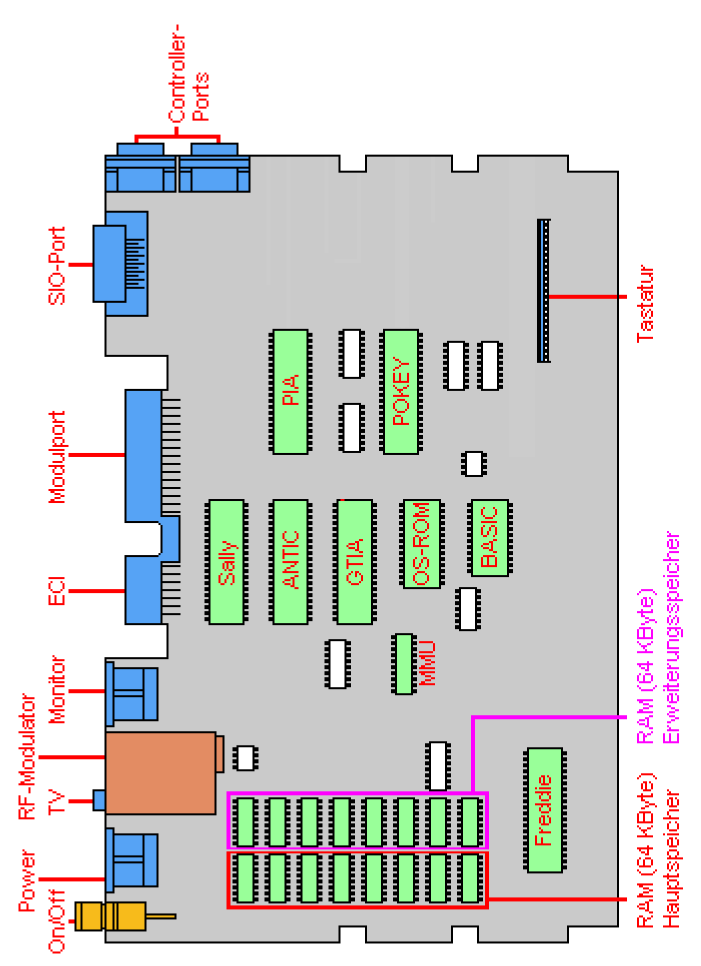


Abb. 4.32: Mainboard XE - Typ 1 mit 8/16 DRAMs 1x64 KBit

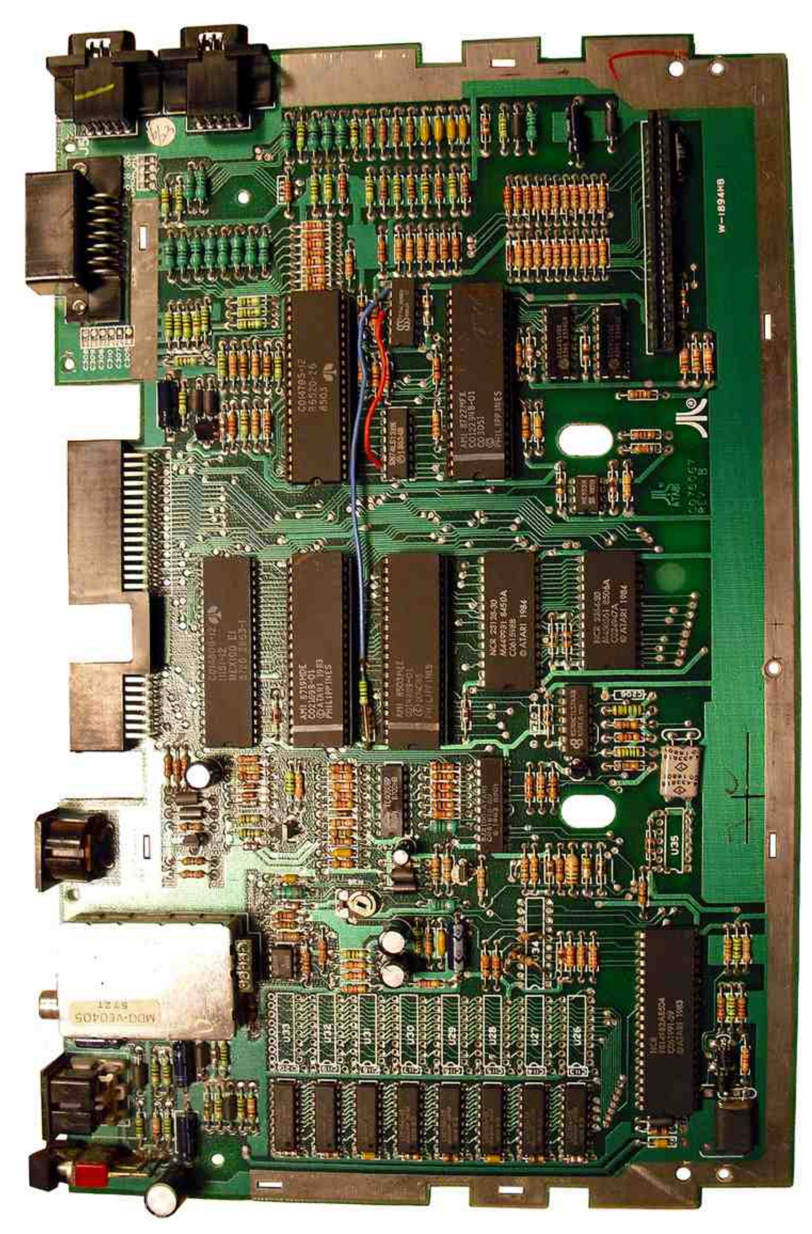


Abb. 4.33: Foto Mainboard XE - Typ 1 mit 8 DRAMs 1x64 KBit

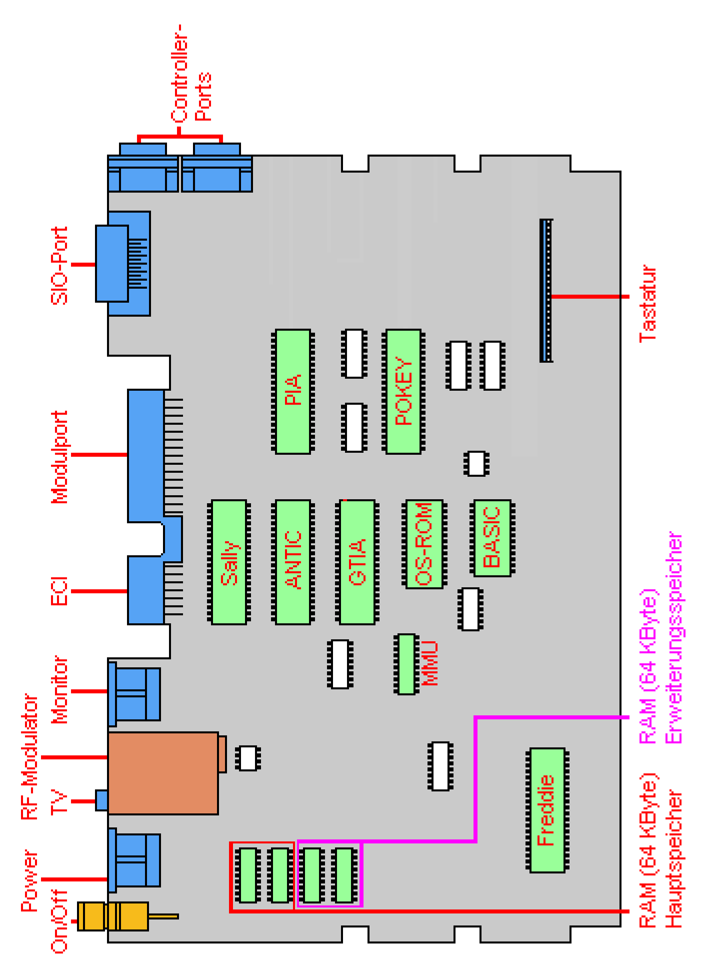


Abb. 4.34: Mainboard XE - Typ 2 mit 2/4 DRAMs 4x64 KBit

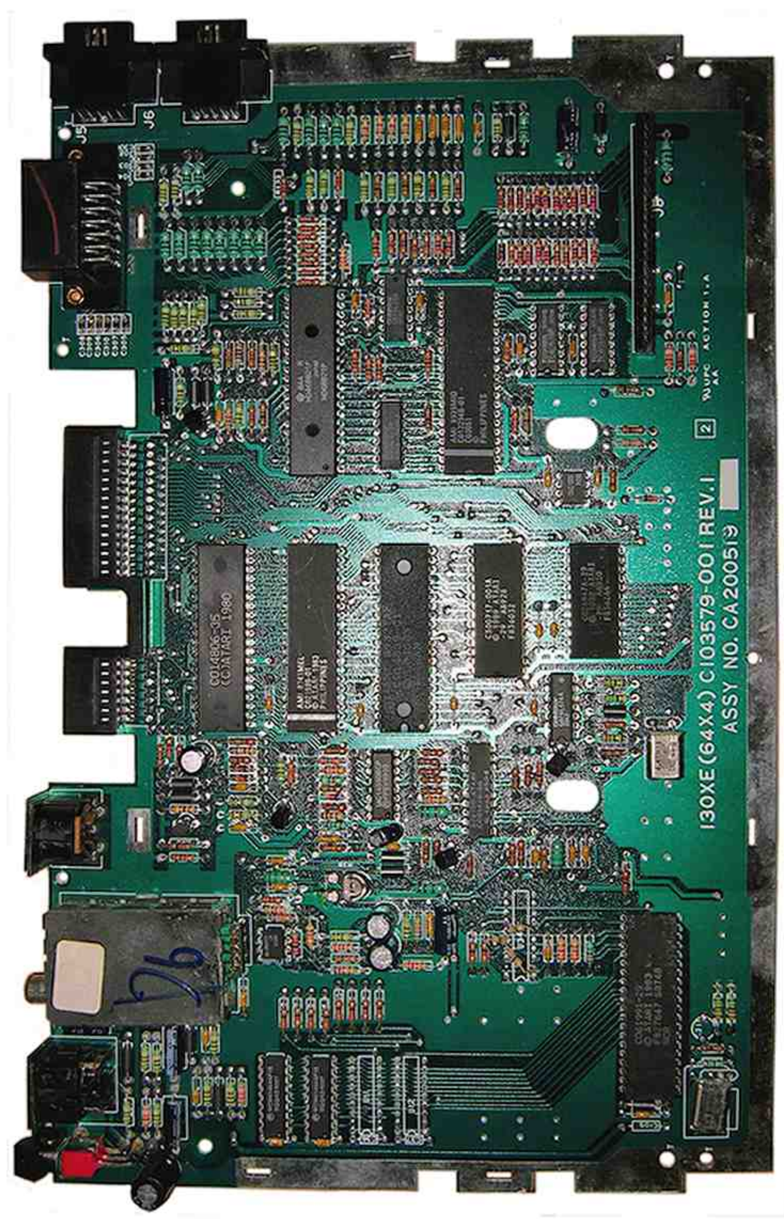


Abb. 4.35: Foto Mainboard XE - Typ 2 mit 2 DRAMs 4x64 KBit

4.16 ATARI XEGS

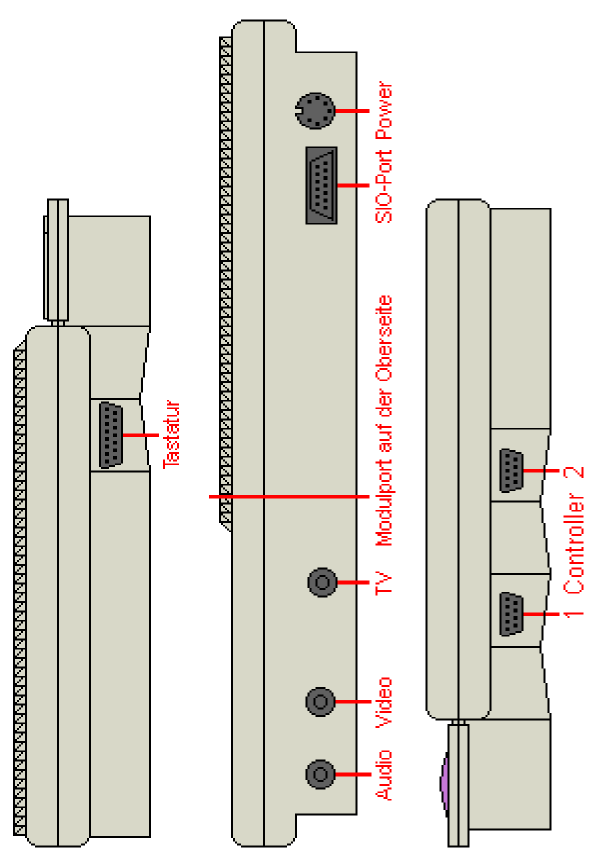


Abb. 4.36: XEGS Anschlüsse links - hinten - rechts

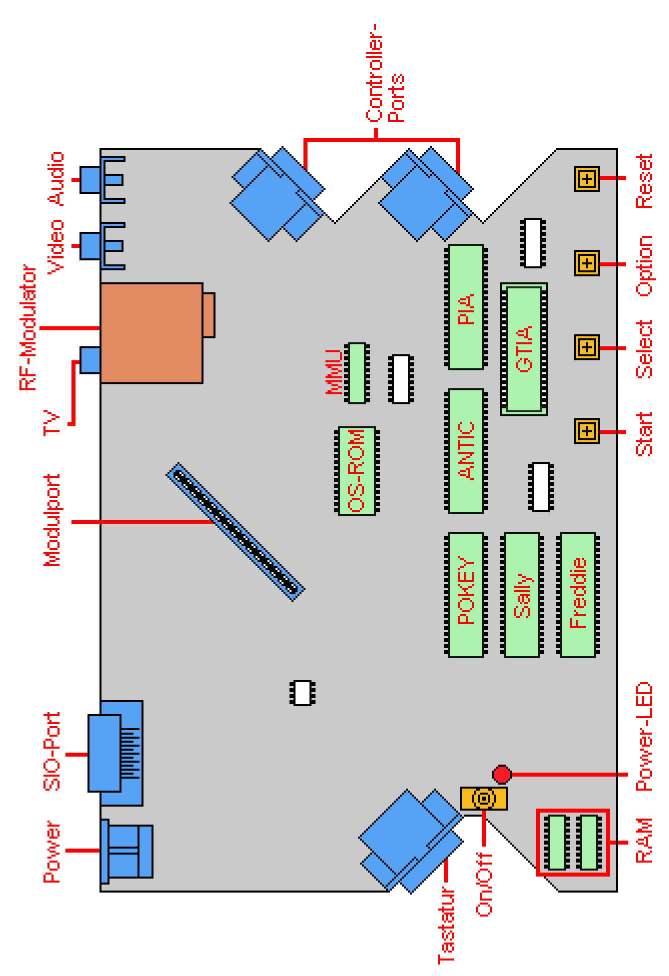


Abb. 4.37: Mainboard XEGS (PAL)

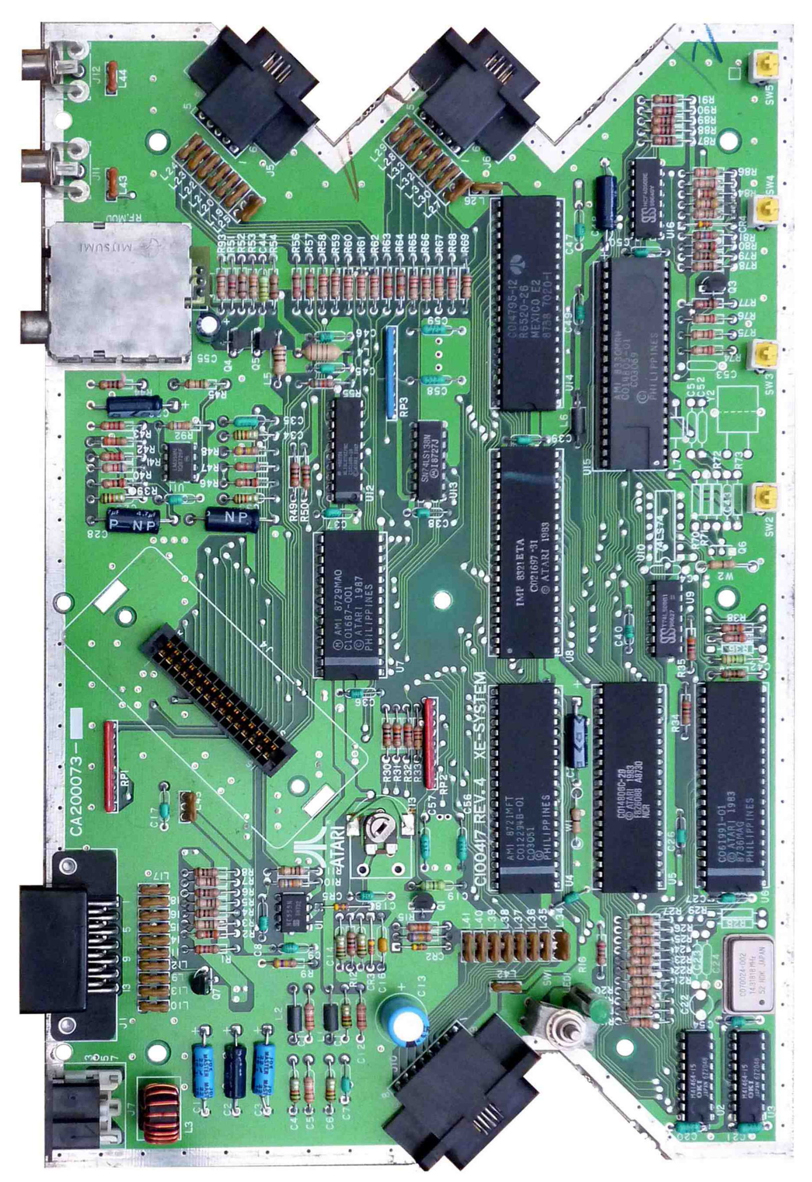


Abb. 4.38: Mainboard XEGS (NTSC)

4.17 Schaltpläne

Auf den in der originalen Ausgabe vorhandenen Schaltplan wurde hier verzichtet, da er aufgrund der Verkleinerung kaum lesbar war und darüber hinaus nur für eine einzige Variante des 800XL gültig ist.

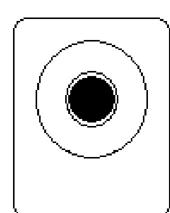
Die Hardware von ATARI unterlag einer ständigen Revision, wodurch sich regelmäßig Änderungen im Layout der Platinen sowie der Bestückung mit Bauteilen ergaben.

Schaltpläne zu den 8-Bit-Computern und den Peripheriegeräten dazu sind über den ABBUC e.V. erhältlich.

4.18 Buchsenbelegungen am 400/800/XL/XE/XEGS

Die Buchsenbelegungen sind jeweils von außen gesehen. Belegungen und Ausführungen sind nicht einheitlich und weisen z.T. wichtige Unterschiede auf.

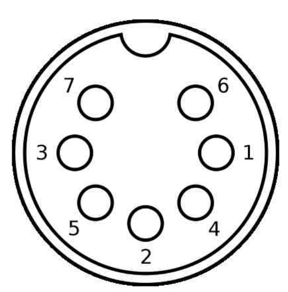
4.18.1 Power 400/800/1200XL



Diese Geräte werden von dem mitgelieferten Trafo mit 9 Volt Wechselspannung gespeist. Die Umwandlung in die benötigte Gleichspannung findet durch die in den Geräten eingebauten Netzteile statt. Der Stromanschluss ist eine Hohlsteckerbuchse für Hohlstecker ohne Innenstift (Øi=2,1mm Øa=5,5mm).

Achtung! Keine Netzteile verwenden, die 9 Volt Gleichspannung liefern.

4.18.2 Power XL/XE/XEGS

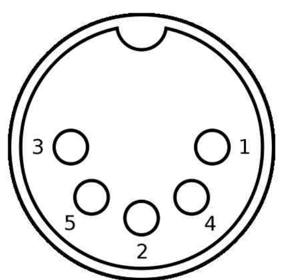


Diese Geräte werden mit 5 Volt Gleichspannung vom mitgelieferten externen Netzteil gespeist.

1,6,4	Eingang +5 Volt
2	Shield (Abschirmung)
3,5,7	Ground (Masse)

Der Stromanschluss ist eine 7-polige DIN-Buchse (270°).

4.18.3 Monitoranschluss 800/XL/XE/XEGS



Alle ATARI 400 und XEGS sowie der 600XL (NTSC) haben keine Monitorbuchse. Die Computer besitzen nur einen HF-Videoausgang. Das XEGS verfügt alternativ über einen AV-Ausgang (2x Cinch).

Bei den XL-Modellen liegen nicht alle Signale an der 5-poligen DIN-Buchse an. Sie lassen sich jedoch leicht im Selbstbau nachrüsten.

Beim 600XL (PAL) ist Pin 5 mit Masse verbunden. Achtet darauf beim Anschluss eines Monitors.

Der Monitoranschluss ist eine 5-polige DIN-Buchse (180°).

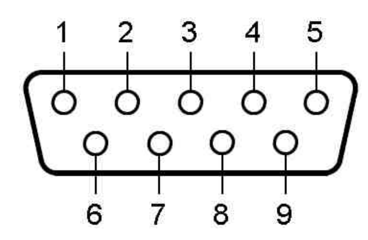
Pin	Signal	für	XL			800/XE
			600XL	800XL	1200XL	
1	Composite Luminance	Einfarb-monitor	-	x	x	x
2	Ground		x	x	x	x
3	Audio Out	NF-Verstärker	x	x	x	x
4	Composite Video (FBAS)	Mehrfarb-monitor	x	x	x	x
5	Composite Chroma (Color + Burst)	Farbmonitor mit sep. Eingängen	- (GND!)	-	-	x

Das Monitorkabel muss zu den Signaleingängen des Monitors passen. Für einen Monochrommonitor, der sich auch bei Verwendung einer XEP80 empfiehlt, statt FBAS das Signal Composite Luminance verwenden. Beim 600XL kann man es selbst nachrüsten.

Die Signalaufteilung in Composite Luminance und Composite Chroma erfordert Farbmonitore, die diese Signale direkt verarbeiten können. Die XL-Modelle lassen sich dafür im Eigenbau entsprechend nachrüsten.

Für die heute oft verwendeten TFT-Fernseher bieten sich das Verbessern der Bildaufbereitung im ATARI und ggf. eine Nachrüstung mit S-Video an.

4.18.4 Controller-Port



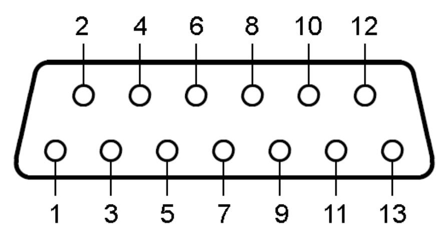
Der Controller-Port ist bi-direktional und kann für Ein- und Ausgaben verwendet werden (→ Joysticks und Paddles).

Beispiel für die Ausgabe ist die → XEP80.

Die Ports sind 9-polige D-Sub-Konnektoren.

1	Eingabe (Joystick vorwärts)	(umschaltbar auf Ausgabe)
2	Eingabe (Joystick rückwärts)	(umschaltbar auf Ausgabe)
3	Eingabe (Joystick links)	(umschaltbar auf Ausgabe)
4	Eingabe (Joystick rechts)	(umschaltbar auf Ausgabe)
5	Eingabe von Potentiometer B	
6	Eingabe (Feuer-) Knopf	
7	+5 Volt (Ausgabe)	
8	Ground (Masse)	
9	Eingabe von Potentiometer A	

4.18.5 SIO-Port



Der serielle Ein-/Ausgabe-Anschluss stellt mehr als nur die Ein-/Ausgabe-Funktionen bereit. So kann z.B. ein Audiosignal eingegeben oder ein Kassettenrecorder gesteuert werden.

1	Clock Input
2	Clock Output
3	Data Input
4	Ground (Masse)
5	Data Output
6	Ground (Masse)
7	Command
8	Motor Control
9	Proceed
10	+5 Volt / Ready
11	Audio Input
12	400/800: +12 Volt XL/XE: nicht belegt
13	Interrupt

Die Verwendung mehrerer ATARIs innerhalb einer SIO-Kette ist möglich, wenn die Ready-Signale der Computer entkoppelt werden.

4.18.6 Parallelbus-Interface 600XL/800XL

Obere Reihe:

1	GND	Ground	27	D6	Data Bus
3	A0	Address Bus	29	GND	Ground
5	A2	Address Bus	31	PHI2	Phase 2 Clock
7	A4	Address Bus	33	N/C	Not connected
9	A6	Address Bus	35	IRQ	Interrupt Request
11	A7	Address Bus	37	N/C	Not connected
13	A9	Address Bus	39	N/C	Not connected
15	A11	Address Bus	41	$\overline{CAS}$	Column Address Strobe
17	A13	Address Bus	43	$\overline{MPD}$	Math Pack Disable
19	GND	Ground	45	GND	Ground
21	D0	Data Bus	47	N/C	Not connected *)
23	D2	Data Bus	49	AUDIO	Audio IN
25	D4	Data Bus			*) 600XL: +5 Volt

1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50

Untere Reihe:

2	$\overline{EXTSEL}$	(RAM)	28	D7	Data Bus
4	A1	Address Bus	30	GND	Ground
6	A3	Address Bus	32	GND	Ground
8	A5	Address Bus	34	$\overline{RST}$	Reset
10	GND	Ground	36	$\overline{RDY}$	Ready
12	A8	Address Bus	38	EXTENB	External Decoder Enable
14	A10	Address Bus	40	$\overline{REFRESH}$	Refresh
16	A12	Address Bus	42	GND	Ground
18	A14	Address Bus	44	$\overline{RAS}$	Row Address Strobe
20	A15	Address Bus	46	LR/$\overline{W}$	Latched Read/Write
22	D1	Data Bus	48	N/C	Not connected *)
24	D3	Data Bus	50	GND	Ground
26	D5	Data Bus			* 600XL: +5 Volt

EXTENB: H → CPU oder ANTIC greifen gerade auf das RAM zu.
EXTSEL: L → deaktiviert das interne RAM des ATARI.

4.18.7 Modulport XL/XE/XEGS - linker Port bei 800

Adressbereich 32768-49151 ($8000-$BFFF)

A	B	C	D	E	F	H	J	K	L	M	N	P	R	S	XE oben	XL vorn
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	unten	hinten

A	RD4	*)	1	$\overline{S4}$	**)
B	GND	Ground	2	A3	Address Bus
C	A4	Address Bus	3	A2	Address Bus
D	A5	Address Bus	4	A1	Address Bus
E	A6	Address Bus	5	A0	Address Bus
F	A7	Address Bus	6	D4	Data Bus
H	A8	Address Bus	7	D5	Data Bus
J	A9	Address Bus	8	D2	Data Bus
K	A12	Address Bus	9	D1	Data Bus
L	D3	Data Bus	10	D0	Data Bus
M	D7	Data Bus	11	D6	Data Bus
N	A11	Address Bus	12	$\overline{S5}$	**)
P	A10	Address Bus	13	+5 Volt	Power
R	R/$\overline{W}$	Read/$\overline{\text{Write}}$	14	RD5	*)
S	PHI2	Phase 2 Clock	15	$\overline{CCTL}$	Cartridge Control

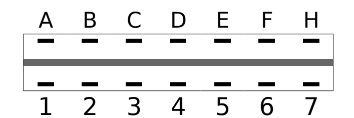
*	RD4 & RD5	ROM Detect for 8K block 4 & 5
**	S4 & S5	Selected Lines 4 & 5 of Memory Map

Der rechte Modulport des 800 ist identisch aufgebaut. Da er aber nur den Adressbereich 32768-40959 ($8000-$9FFF) abdeckt, sind die Anschlüsse 12 und 14 nicht belegt.

RD4	vom Modulport: H=Modul will Speicher (meist ROM) bei $8000-$9FFF einblenden.
RD5	vom Modulport: H=Modul will Speicher (meist ROM) bei $A000-$BFFF einblenden.
$\overline{S4}$	zum Modulport: L=CPU greift auf $8000-$9FFF zu und RD4 ist H, d.h., Modul soll seinen Speicher (ROM) aktivieren.
$\overline{S5}$	zum Modulport: L=CPU greift auf $A000-$BFFF zu und RD5 ist H, d.h., Modul soll seinen Speicher (ROM) aktivieren.

4.18.8 Enhanced Cartridge Interface XE (ECI)

Das beim 600XL und 800XL vorhandene Parallelbus-Interface findet ihr bei den XE-Modellen nur in einer Sparversion vor. Die in Deutschland vermarkteten XE-Computer (PAL) verfügen alle über ein ECI.



Pin	Signal	Beschreibung	Pin	Signal	Beschreibung
A	N/C	Not connected	1	$\overline{\text{EXTSEL}}$	(RAM)
B	$\overline{\text{IRQ}}$	Interrupt Request	2	$\overline{\text{RST}}$	Reset
C	$\overline{\text{HALT}}$	Halt	3	$\overline{\text{D1xx}}$	PBI device I/O
D	A13	Address Bus	4	$\overline{\text{MPD}}$	Math pack Disable
E	A14	Address Bus	5	Audio	Audio IN
F	A15	Address Bus	6	$\overline{\text{REF}}$	Refresh
H	GND	Ground	7	+5 Volt	

Bei der NTSC-Version des 65XE ist der ECI nicht nach außen geführt.

PBI-Geräte für den XL können über einen passenden XE-Adapter angeschlossen werden.

Anhang

Quellenverzeichnis

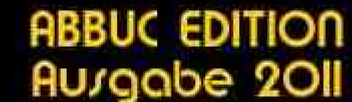
ATARI Basic-Handbuch (SYBEX, Best.-Nr. 3083)
De Re Atari (ATARI Inc.)
INSIDE ATARI BASIC (Compute Books)
OS-Listing (ATARI Inc.)
OS-Manual (ATARI Inc.)
Schaltpläne der A8-Computer
Kommentiertes OS-Listing des XL
Programmierung des 6502 (SYBEX, Best.-Nr. 3011)
Hardware-Manual (ATARI Inc.)
Mapping the ATARI (Compute Books)
Master Memory Map For The ATARI (Reston Computer Books)
Your ATARI Computer (Osborne/McGraw-Hill)
ATARI 130XE Handbuch (ATARI Corp.)
ATARI 600XL/800XL Handbuch (ATARI Corp.)
MyDOS Dokumentation von Robert Puff (1988)
SpartaDOS Construction Set Manual (ICD)
SpartaDOS X V. 4.1x – 4.2x Manual (ICD)
BeweDOS Manual
SpartaDOS X V. 4.4x Manual (DLT)
RealDOS Manual (ILS)
www.atarimuseum.com (Entwicklerunterlagen)
BYTE magazine, September 1985 (The QuarterMeg)
Happy Computer, Ausgabe 12/1985 (Turbo BASIC 1.5 XL)
http://www.pcworld.com/article/181421/inside_the_atari_800.html

Abbildungsverzeichnis

Tabellenverzeichnis

Stichwortverzeichnis

SYBEX



Das ATARI Profibuch

Als Besitzer eines **ATARI 400, 800, 1200XL, 600XL, 800XL, 65XE, 130XE, 800XE oder eines XE Game System** hast Du jetzt das Buch in der Hand, dessen Platz neben dem Rechner ist. Es enthält gebündelt die wichtigsten Informationen über Deinen ATARI Computer und ermöglicht Dir, seine Fähigkeiten voll auszureizen.

Das ATARI Profibuch bietet dem interessierten Nutzer

- eine vollständige Liste aller Systemadressen und -routinen mit Erläuterungen,
- detaillierte Informationen über Speicherbelegung und Nutzung des Betriebssystems,
- einen Tabellenteil mit den Systemcodes, 6502-Opcodes und einer alphabetischen Liste aller Systemadressen,
- viele weitere nützliche Tabellen im gesamten Text,
- Informationen zu verschiedenen DOS, verschiedenen Cartridge-Typen und dem Parallelbus-Interface,
- Ein komplett neu erarbeitetes Kapitel 4 zu den Anschlüssen am und im Computer und seinen wichtigsten Chips mit vielen Erläuterungen und Fotos.

Dein Standardwerk zu Deinem ATARI-8-Bit-Computer!

Die Autoren Julian Reschke & Andreas Wiethoff sowie der Sybex-Verlag ermöglichten durch die Freigabe der Inhalte dem ABBUC e.V. die aktualisierte Ausgabe 2011.



ABBUC
EDITION
2011

Reschke
Wiethoff

Das ATARI Profibuch



